**MASTER EN SCIENCES INFORMATIQUES**

# Un générateur de dialogues sous MS-Windows

Bouhon, François

*Award date:*
1988

*Awarding institution:*
Universite de Namur

# UN GENERATEUR

# DE DIALOGUES

# SOUS MS-WINDOWS.

François Bochon
3e Licence Informatique

Année Académique 1987-1988

# UN GENERATEUR DE DIALOGUES SOUS MS-WINDOWS.

François Bouhon
3° Licence Informatique

Résumé.

Comment concevoir et implémenter une interface conviviale et ergonomique pour un langage de commande dans l'environnement Windows de Microsoft? Comment automatiser la production d'une telle interface? Voilà les questions auxquelles nous avons essayé d'apporter une réponse par un exemple concret, dans lequel les commandes du langage ont été transformées en simples formulaires à remplir au gré de l'utilisateur. Dans ce mémoire, nous retraçons les différentes étapes suivies pour atteindre cet objectif.

Abstract.

How to conceive and implement a friendly and ergonomic user interface for a command language under the Microsoft Windows environment? How to generate automatically such an interface? These are the questions to which we tried to bring an answer with a concrete example, where the command language has been transformed into simple forms which the user may fill in the way he wants to. In this dissertation, we give a short view of the different steps we have followed to achieve this goal.

Nous tenons ici à exprimer toute notre reconnaissance à Mr Bodart, notre promoteur de mémoire, à Mr Foucart et aux autres membres du projet IDA pour l'aide efficace qu'ils nous ont apportée tout au long de ce mémoire. ainsi que Mr Bertram et ses collaborateurs avec qui nous avons eu le privilège d'effectuer notre stage.

Nos plus sincères remerciements également à toutes les personnes qui ont participé à la réalisation de ce mémoire.

PLAN.

# INTRODUCTION.

DSL-STATION est un poste de travail, spécialement conçu pour l'environnement MS-Windows, et qui fait partie du logiciel de spécifications IDA. Ce poste de travail permet à ses utilisateurs de travailler de manière décentralisée par rapport à la machine princiale sur laquelle tourne le reste du logiciel et qui contient toutes les bases de données de spécifications de l'application en développement. DSL-STATION permet de travailler soit en interaction avec le site central, par l'intermédiaire d'un émulateur de terminal et d'un réseau, soit en mode autonome, sur des bases de données locales. Dans ce cas, l'autonomie du poste de travail est limitée par la nécessité d'une interaction avec le site central lors du garnissage initial des bases locales et, une fois la tâche terminée, pour y répercuter les modifications apportées.

Le poste de travail se compose de plusieurs programmes, à savoir un éditeur de langage DSL, un éditeur de texte libre et un éditeur de commandes DSL-SPEC. L'éditeur de langage permet la définition des spécifications à introduire dans la base de données, éventuellement en interaction avec l'éditeur de texte libre. Quant à l'éditeur de commandes, il a pour but la définition des commandes de manipulation de la base de spécifications. Ces commandes se divisent en plusieurs groupes, utilisés respectivement pour la gestion de l'environnement du système, la modification ou l'interrogation du contenu de la base de données, ou encore la production de rapports documentaires sur celle-ci.

C'est à ce dernier éditeur que nous nous intéresserons tout au long du présent mémoire.

Bien qu'ayant été spécifiquement conçu pour tourner dans l'environnement Windows, l'éditeur de commandes n'en utilise les principaux avantages que de façon assez limitée. C'est pourquoi, à notre retour de stage, où nous avions appris à nous familiariser avec cet environnement, il nous a été demandé de réviser fondamentalement l'interface existante, afin de la rendre plus conviviale, plus ergonomique. Notre travail a consisté non seulement à formuler un certain nombre de critiques sur cette interface et à élaborer une série de propositions visant à résoudre les problèmes de convivialité qu'elle engendrait, mais également à les implémenter. Nous nous proposons, dans ce mémoire, d'en relater les différentes étapes.

# CHAPITRE I:

# POINT DE DEPART DU PROJET

# 1. INTRODUCTION.

Comme pour tout projet s'inscrivant dans un contexte logiciel déjà établi, il nous faudra respecter un certain nombre de contraintes dépendant de manière plus ou moins directe de ce contexte. Nous les exposerons dans la première partie de ce chapitre.

Nous présenterons alors une analyse critique de l'interface actuelle de DSL-SPEC tournant dans l'environnement MS-Windows dans le cadre du poste de travail DSL-Station. Les critères sur lesquels nous avons fondé notre appréciation sont essentiellement d'ordre ergonomique, mais nous avons également tenté de tirer un maximum de profit de notre connaissance de cet environnement et des programmes qu'il supporte, connaissance acquise à l'occasion de notre stage, afin de veiller à ce que les modifications que nous avons proposées s'insèrent assez harmonieusement dans les standards couramment utilisés.

Nous aborderons finalement ce qui constitue la base même sur laquelle repose cette interface, à savoir la description syntaxique du langage de commande.

## 2. CONTRAINTES.

Il va de soi que la modification d'un outil intégré dans un logiciel tel que IDA, ne peut se concevoir sans un certain nombre de contraintes. Celles-ci apparaissent souvent comme une limitation dans la définition d'une interface "idéale". La première et la plus évidente de ces contraintes est le fait que, quel que soit le mode de représentation du langage de commandes, il est impératif que les résultats produits demeurent acceptables pour l'interpréteur auquel ils sont destinés.

C'est pourquoi, même si l'interface proposée parvient à masquer certaines incohérences du langage, celles-ci se retrouveront tant dans l'utilisation de l'éditeur de commandes en mode expert que dans la commande destinée à l'interpréteur.

D'autre part, il convient de préciser que la description syntaxique du langage, sur laquelle se base le générateur de dialogues, n'est pas destinée qu'à lui seul, mais sert également sur le site central. Les modifications que nous pouvons y apporter sont donc fort limitées par ce double usage.

Finalement, et c'est la raison d'être du générateur de dialogues, l'interface doit être produite de manière automatique et généralisable à d'autres langages de commande similaires.

## 2. ANALYSE CRITIQUE.

### A. Présentation de l'éditeur de commandes.

L'interface, telle qu'elle était implémentée à l'époque où nous avons entrepris ce travail, paraît, à première vue, relativement simple. L'éditeur de commandes comporte deux types d'écrans, différenciés par le type d'informations qui apparaissent dans la fenêtre principale de l'application: d'une part ce que nous appellerons désormais l'écran d'accueil, où sont affichées toutes les commandes du langage, et d'autre part les écrans des commandes, qui contiennent les paramètres de chaque commande ainsi que les valeurs attribuées à ces paramètres.

Chaque écran comporte deux fenêtres d'état, l'une indiquant quel est le mode de fonctionnement actuel (interactif ou batch) et l'autre mentionnant soit le nom de la commande en cours d'édition, soit CLI (pour Command Language Interface) dans le cas de l'écran d'accueil. Le reste de la zone cliente est occupé par la liste des commandes ou des paramètres (voir Figures 1 et 2).

### B. Utilisation des menus.

Un premier critère à retenir dans notre évaluation de l'éditeur de commandes concerne les menus, puisqu'il s'en trouve, sous différentes formes, à tous les niveaux de l'interface. Ces menus se distinguent tant par leur organisation sémantique, dont Shneiderman décrit de nombreuses possibilités (Shneiderman,

pp. 87-105) en insistant sur l'aspect hiérarchisable du problème, que par la manière dont ils sont représentés dans l'interface qui nous occupe. Un second critère, relatif à la répartition des données sur l'écran, sera également abordé, avant d'être étudié de manière plus précise dans le cas particulier des écrans des commandes.

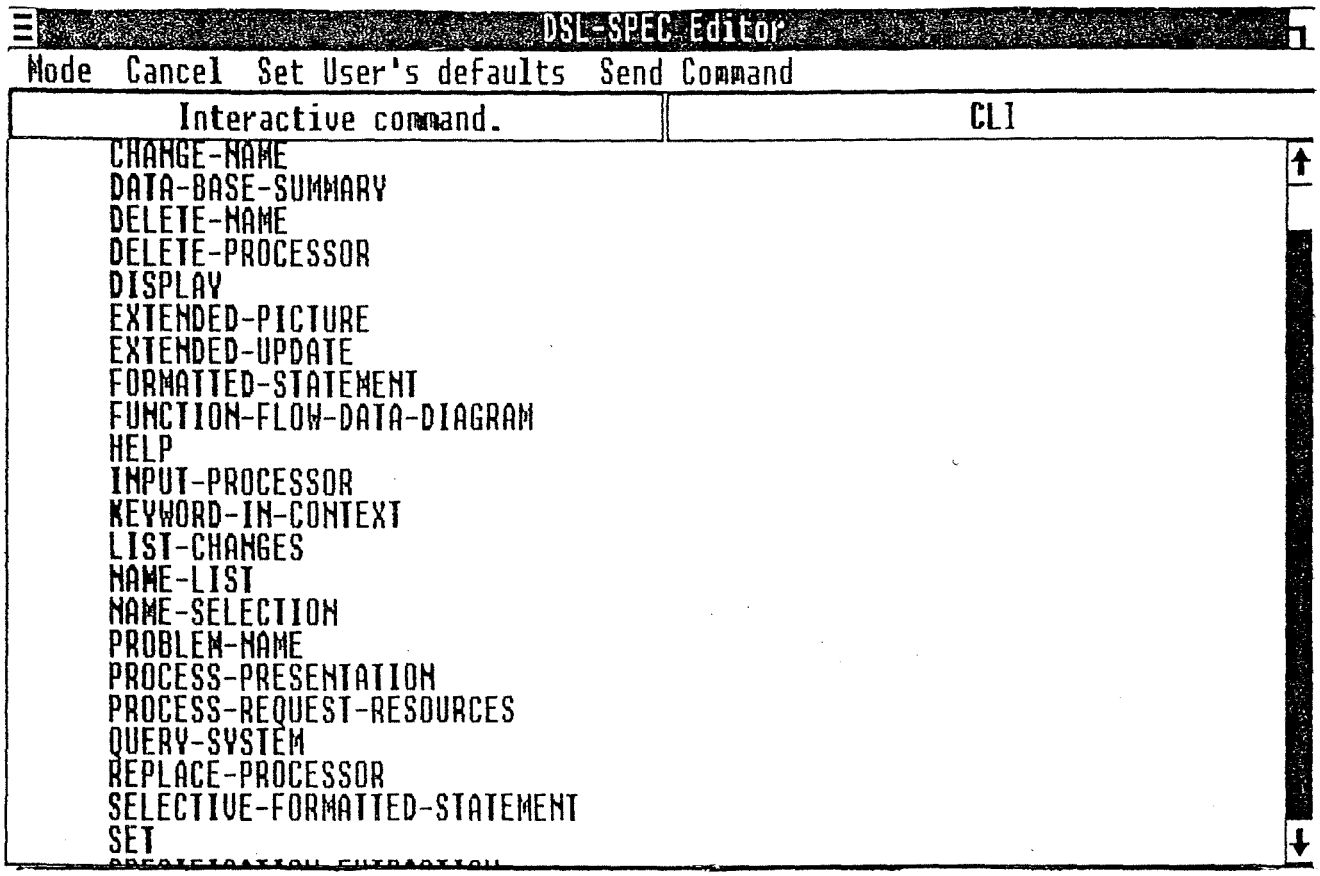



Figure 1.

a. Les commandes du langage.

Tout d'abord, la liste des commandes peut être considérée comme le menu de niveau supérieur: il faut d'abord sélectionner une commande avant de pouvoir accéder à l'édition proprement dite. Cette liste est présentée comme une suite de noms de commandes disposés les uns en-dessous des autres dans la zone cliente de l'éditeur (voir Figure 1). Cependant, étant donné la longueur de cette liste, qui compte près de trente commandes différentes, elle ne peut être affichée que partiellement, en fonction de la taille de la fenêtre.

Ce problème est résolu grâce à une barre de défilement verticale. Mais son utilisation présente un inconvénient majeur, à savoir le fait que le seul défilement ligne par ligne et à l'aide de la souris soit implémenté. De plus, chaque fois que la liste des commandes doit être réaffichée, par exemple après que l'utilisateur ait terminé l'édition d'une commande, l'affichage se fait à partir du début de la liste. Cet aspect, pouvant passer pour un simple détail, se révèle particulièrement gênant lorsque, comme c'est le cas, la liste est assez longue ou que la hauteur de la fenêtre est réduite, ce qui se produit dans le cas très fréquent où deux ou plusieurs applications se partagent l'écran.

Si cette représentation est simple et uniforme, elle peut quand même laisser perplexe l'utilisateur qui aborde l'éditeur pour la première fois et qui ne connait pas grand chose au langage

de commande: à quoi peuvent bien servir toutes ces commandes, et comment les distinguer? D'aucuns regretteront sans doute que cette liste ne soit pas structurée, hiérarchisée, ce qui en rendrait l'utilisation beaucoup plus simple, particulièrement pour l'utilisateur novice.

De plus, il y a gros à parier que même un utilisateur habituel du langage se limitera à quelques commandes qu'il aura appris à bien connaître, les autres ne servant qu'exceptionnellement voire pas du tout. Il en va ainsi de la plupart des langages de commande. Cet aspect est pris en compte dans l'ordre d'apparition des commandes à l'écran: il a été établi en fonction de leur importance et de leur fréquence d'utilisation.

b. Les paramètres des commandes.

Un second niveau de menu est constitué par les listes de paramètres dans les écrans des commandes. Il se différencie du premier en ce sens qu'il ne s'agit plus de sélectionner un seul item, la commande à éditer ou à envoyer, mais un nombre variable d'options à affecter aux commandes. Tout comme ces dernières, les paramètres sont affichés les uns en-dessous des autres, et leur ordre d'apparition à l'écran a été établi selon les mêmes critères (voir Figure 2). Les objections et remarques que nous avons formulées à propos de la liste des commandes restent donc valables dans le cadre des paramètres, bien que leur importance respective ne soit pas toujours équivalente. Par exemple, certaines listes de paramètres sont assez courtes et toutes les informations qu'elles

contiennent peuvent aisément être visibles même dans une fenêtre de taille réduite. Cependant, il peut se faire que des paramètres interdépendants n'apparaissent pas en même temps, ce qui est regrettable. D'autre part, il paraît déjà moins évident de tenter de hiérarchiser les listes de paramètres.

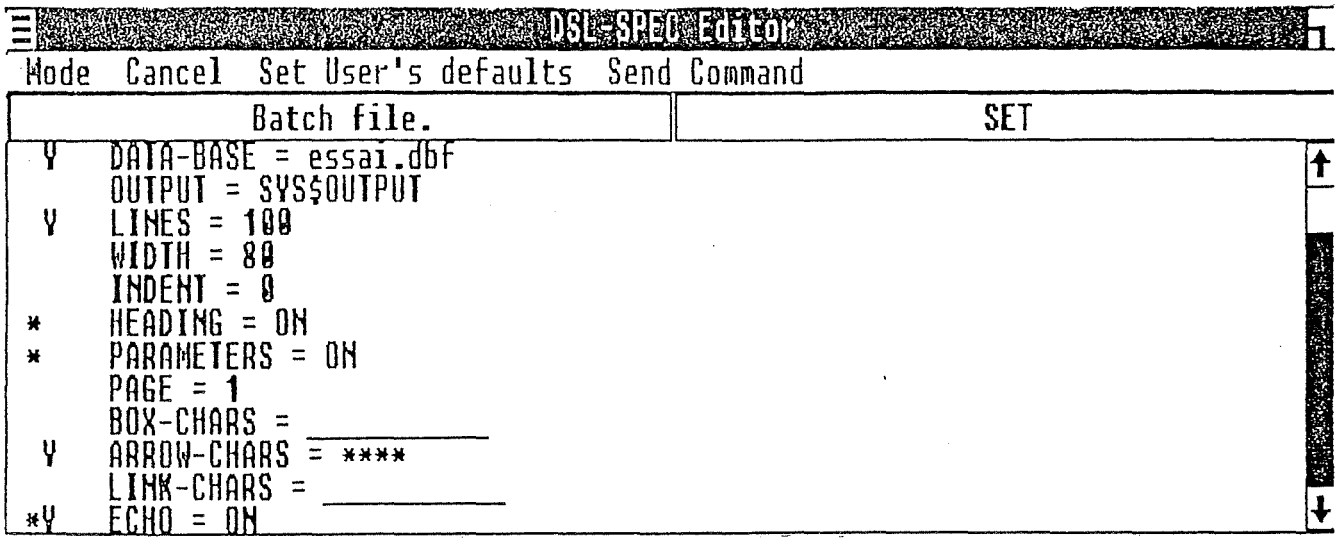



Figure 2.

c. Les valeurs des paramètres.

Nous pourrions poursuivre notre étude des menus de l'éditeur de commandes en remarquant que les différentes valeurs prédéfinies que peuvent prendre certains paramètres constituent, elles aussi, des items de menu. Mais ici, seule la valeur actuellement sélectionnée apparaît, les autres étant accessibles à tour de rôle par des double-clics. Il est donc impossible de visualiser en une fois ces différentes valeurs (voir Figure 2).

d. La barre de menu.

Enfin, un menu de commandes spécifiques à l'interface est également proposé au sein d'une barre de menu accessible à tout moment, qui permet de choisir le mode de fonctionnement de l'éditeur de commandes (interactif ou batch), d'annuler une ligne ou une commande, de mémoriser les valeurs par défaut de l'utilisateur et d'envoyer une commande à l'émulateur de terminal ou dans un fichier batch, selon le mode de fonctionnement choisi (Figure 1). Remarquons au passage que ces options se rapportent soit à l'écran d'accueil, soit aux écrans des commandes, et que le fait de les voir associées dans une même barre de menu risque sans doute de prêter à confusion. Il aurait sans doute mieux valu adopter une présentation de ces commandes plus proche de la tâche de l'utilisateur, et suivre la décomposition entre l'écran d'accueil et ceux des commandes.

De plus, s'il est permis à l'utilisateur de mémoriser ses propres valeurs par défaut, aucune option ne lui permet de s'en servir par la suite. Il y a donc là apparemment une lacune dans les fonctionnalités de l'éditeur de commandes.

C. Informations dans les écrans des commandes.

Comme nous venons de l'exposer, les paramètres figurant dans les écrans des commandes sont disposés verticalement. Chacun d'eux est entouré de divers champs d'information que nous allons quelque peu détailler (voir Figure 2).

Les paramètres sont tous affichés avec la valeur par défaut que le système leur reconnaît, s'il y en a une. Le nom des paramètres pouvant prendre plusieurs valeurs prédéfinies et ne nécessitant pas d'entrée de valeur est précédé d'un signe particulier (*).

Lorsqu'il n'existe pas de valeur prédéfinie pour un paramètre, la valeur qu'il prendra doit être introduite par l'utilisateur. Elle apparaîtra alors, elle aussi, à la suite du paramètre. L'absence de valeur est signalée par une ligne de tirets (_) derrière le nom du paramètre.

Pour indiquer à l'utilisateur qu'il a déjà modifié la valeur d'un paramètre, la lettre Y est placée devant son nom. Enfin, un troisième signe particulier (#) apparaissant en préfixe du paramètre sert à marquer l'obligation de lui attribuer une valeur pour que la commande puisse être validée.

Cela nous amène à considérer d'autres critères d'évaluation de l'interface, qui ont trait à la conception même des écrans. Ces unités de mesure de la complexité des écrans sont décrites dans (Shneiderman, pp. 331-335).

S'il est vrai que la faible densité globale des informations intervenant au cours de l'édition des commandes, mesurée par rapport aux espaces vierges des écrans, et leur regroupement autour des paramètres contribuent à leur lisibilité, et par là à la facilité d'utilisation de l'éditeur de commandes, par contre la densité locale de ces informations, considérée dans un champ de

vision restreint, est assez élevée, puisqu'elles sont toutes rassemblées dans la même partie des écrans.

Il apparaît donc immédiatement que l'espace disponible sur l'écran est sous-utilisé, ou plus exactement que la répartition des informations y est fort inégale. En effet, la moitié droite de la zone où se trouve la liste des commandes et des paramètres ne sera utilisée que pour quelques paramètres acceptant une valeur assez longue. D'autre part, la taille des listes de paramètres et de commandes excède souvent la place disponible, rendant obligatoire le défilement vertical de la fenêtre où elles apparaissent.

D. Entrées de valeurs.

Un autre point pouvant être amélioré, à notre avis, est la manière dont sont réalisées les entrées de valeurs, que nous évaluerons en fonction de critères retenus par Shneiderman (pp.72-73), en particulier les actions que l'utilisateur doit effectuer. Nous ferons également part de quelques remarques plus personnelles, en rapport avec l'environnement Windows.

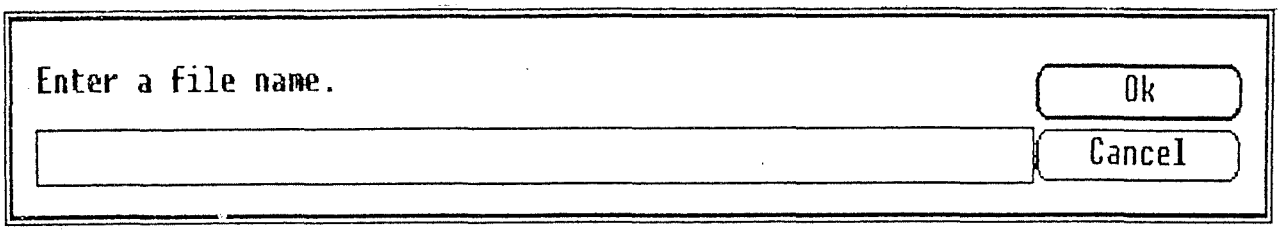



Figure 3.

Les entrées de valeurs se font toutes au moyen de la même boîte de dialogue (voir Figure 3), paramétrée selon le type d'entrée: elle est composée d'une ligne de texte indiquant le type de valeur attendu et éventuellement sa taille maximale (chaîne de caractères), ses bornes (entiers) ou le nombre maximal d'objets (liste de noms), d'une fenêtre d'édition où la valeur sera introduite, et de deux boutons poussoirs (Ok et Cancel) permettant de faire valider la valeur introduite ou d'annuler la réponse. Après un contrôle syntaxique, celle-ci est affichée sur l'écran à côté du nom du paramètre.

Bien qu'il soit aisé d'apprendre la signification de ces boîtes de dialogue et de s'en servir, et que, en outre, leur implémentation soit simple à réaliser, il est permis de se demander si cette présentation n'est pas trop standardisée: en effet, seul le texte affiché permet de distinguer les différents types de valeurs attendus. De plus, il est permis de s'interroger quant à l'utilité d'une boîte de dialogue aussi large (elle s'étend sur toute la largeur de l'écran) pour introduire une valeur entière ou une chaîne de quelques caractères.

Le passage par une boîte de dialogue intermédiaire peut sembler excessif pour introduire des valeurs relativement courtes. Par contre, cela permet non seulement de fournir à l'utilisateur quelques explications relatives à la valeur qu'il va introduire, mais également de procéder à une validation syntaxique immédiate

de la valeur introduite, ce qui offre des avantages non négligeables.

E. Traitement des erreurs.

Les erreurs éventuelles sont signalées à l'utilisateur par des boîtes de message ("message boxes"), ce qui est, à notre avis, le meilleur moyen. Toutefois, le contrôle ne se fait qu'après la fermeture de la boîte de dialogue et la valeur introduite est alors irrémédiablement perdue.

Bien que la manière dont sont réalisées les entrées de valeurs favorise la prévention des erreurs, comme le préconise Shneiderman, un point faible de cette interface est la possibilité de les corriger. En effet, en cas d'erreur, il faut réintroduire la valeur en entier, même si un seul caractère était erronné. De même, les messages d'erreur ne sont pas toujours très compréhensibles et ne disent pas clairement ce qu'il faut faire pour réparer l'erreur commise.

F. Niveaux d'utilisation.

Signalons enfin la possibilité d'introduire directement une commande sans passer par les étapes de sélection et de positionnement des paramètres: dans l'écran d'accueil, il suffit de double-cliquer sur le bouton droit de la souris et une boîte de dialogue apparaît, invitant l'utilisateur à introduire une commande et ses paramètres. Cela équivaut à l'introduire directement

dans la fenêtre de l'émulateur de terminal. Nous pensons qu'une telle facilité, particulièrement utile pour des utilisateurs fréquents de l'interface et qui savent exactement à quoi correspondent les commandes et leurs paramètres, doit être conservée. Elle apparaît comme particulièrement importante dans la mesure où l'éditeur de commandes peut être utilisé aussi bien par des experts que par des novices en la matière.

## 4. La syntaxe du langage de commande.

Avant de clôturer cette présentation de l'éditeur de commandes, il est bon de parler brièvement de la syntaxe du langage qu'il supporte. Notre but ici n'est nullemenet de critiquer le langage en lui-même, mais plutôt d'en décrire les principales caractéristiques auxquelles nous ferons fréquement référence dans la suite de ce mémoire.

Chaque commande correspond à une action sur la base de données des spécifications ou sur l'environnement du système. Elle peut être modulée selon différentes options definies par ses paramètres.

Toutes les commandes commencent par un nom de commande optionnellement suivi d'un ou plusieurs paramètres, par exemple:

DISPLAY DATA-BASE LANGUAGE

Apparemment, les formes que peuvent prendre les parametres sont peu nombreuses et assez explicites: soit le nom du paramètre est employé seul, soit une valeur lui est attribuée et il prend alors la forme 'paramètre=valeur' comme dans l'exemple suivant:

SET OUTPUT=SYS$OUTPUT

En fait, le langage est beaucoup plus complexe que cela, pour deux raisons. D'abord la frontière entre ces deux formes génériques ne permet pas de diviser les parametres en deux classes distinctes, certains d'entre eux pouvant selon le cas prendre

l'une ou l'autre forme. Ensuite, l'interpréteur du langage considère certaines options fixées par défaut.

L'emploi d'un paramètre sans qu'aucune valeur lui soit attribuée peut signifier plusieurs choses. D'abord, si l'interpréteur ne connaît pas de valeur par défaut relative à ce paramètre, il s'agit simplement d'ajouter une option à la commande, comme c'est le cas dans l'exemple ci-dessus. D'autre part, le paramètre peut être un booléen; dans ce cas, l'absence ou la présence du préfixe NO devant son nom indique s'il doit être pris en compte ou non. Enfin, deux paramètres de cette forme peuvent être mutuellement exclusifs; l'emploi de l'un ou de l'autre indiquera alors quelle variante d'une même option doit être utilisée.

En ce qui concerne les paramètres auxquels est attribuée une valeur, ils se répartissent également en plusieurs catégories selon que cette valeur est prédéfinie ou propre à l'utilisateur. Certains d'entre eux n'acceptent qu'une valeur prédéfinie parmi deux ou plusieurs possibles; quelques uns n'ont pas de valeur prédéfinie qui leur soit associée; d'autres enfin peuvent admettre l'une ou l'autre sorte de valeur. Ici également, l'interpréteur reconnaît à certains paramètres une valeur par défaut.

La valeur que l'utilisateur peut attribuer à un paramètre sera, selon le cas, un nom de fichier, une chaîne de caractères, un entier, un nom ou une liste de noms d'objets DSL. Ces valeurs doivent respecter certaines règles syntaxiques qu'il n'est pas nécessaire de préciser ici.

Signalons encore que la plupart des commandes, des paramètres et de leurs valeurs prédéfinies peuvent être invoqués sous une forme abrégée. C'est sous cette forme abrégée que les commandes sont transmises au site central ou écrites dans un fichier par l'éditeur de commandes.

# CHAPITRE II :

# CONCEPTION D'UNE NOUVELLE INTERFACE

# 1. INTRODUCTION.

La démarche que nous avons suivie lors de cette étape cruciale de notre travail a été la suivante. Après avoir étudié la description syntaxique du langage et nous être quelque peu familiarisé avec l'interface que nous devions améliorer, nous nous sommes attaqué, dans un premier temps, au problème de la représentation des écrans de commandes. Il nous a en effet semblé que cet aspect de l'interface était, si pas le plus important, du moins le plus susceptible d'être fondamentalement modifié par une utilisation judicieuse des nombreuses possibilités offertes par l'environnement MS-Windows.

Les autres modifications que nous avons jugé utile et intéressant d'apporter à l'interface n'en sont pas moins importantes, mais elles marquent une différence moins tranchée par rapport à l'éditeur de commandes tel qu'il était. Ces propositions constituent la plus importante partie de ce chapitre.

Une fois nos propositions clairement formulées, nous avons rencontré quelques personnes utilisant plus ou moins fréquemment cet éditeur, ou du moins le connaissant de manière relativement complète. Ces entretiens nous ont permis d'arbitrer certains choix de conception et de préciser certains points restés flous. Ils nous ont également apporté quelques suggestions fort intéressantes. Nous en reparlerons avant de clôturer le présent chapitre.

## 2. PROFIL DE L'UTILISATEUR.

Afin de nous guider dans l'élaboration d'une interface à la fois conviviale et efficace, il nous semble nécessaire de brosser un rapide portrait de ceux qui seront amenés à l'utiliser.

La tâche supportée par le poste de travail, et l'éditeur de commandes en particulier, à savoir l'édition et la modification de spécifications, étant bien délimitée, il est clair que ses utilisateurs doivent avoir une bonne connaissance sémantique tant du travail qu'ils accomplissent que des principaux concepts informatiques qui y sont liés.

Par contre, l'éditeur de commandes est de toute évidence destiné à réduire la complexité syntaxique du langage dans des proportions considérables. En outre, notre expérience personnelle nous permet de considérer l'utilisation de l'environnement Windows comme relativement simple à apprendre, du moins dans ses fonctionnalités essentielles. Les connaissances syntaxiques minimales requises nous paraissent donc suffisamment restreintes pour que l'éditeur de commandes DSL-SPEC s'adresse à un éventail d'utilisateurs assez large. C'est dans ce sens que nous parlerons d'utilisateur novice ou expert.

En résumé, nous pouvons considérer l'utilisateur moyen de l'éditeur de commandes DSL-SPEC dans l'environnement Windows comme un "knowledgeable intermittent user", pour reprendre le terme employé par Shneiderman (p. 54).

## 3. MODIFICATIONS PROPOSEES.

Le but des modifications présentées ci-dessous est de rendre l'interface actuelle du langage de commandes de DSL-SPEC sous Windows à la fois plus agréable et plus facile pour l'utilisateur novice, tout en lui permettant de s'en servir de manière plus directe et plus puissante au fur et à mesure que sa maîtrise du langage grandit.

Nous envisagerons successivement, en rapport avec les critiques formulées au cours de la première partie de ce mémoire, la présentation de l'écran d'accueil et des écrans des commandes, l'utilisation respective de la souris et du clavier et le traitement des erreurs. Nous terminerons ces quelques propositions en avançant quelques suggestions destinées à accroître les fonctionnalités de l'éditeur de commandes de manière à le rendre plus ergonomique. Il s'agit de l'addition d'une fonction d'aide et d'une fenêtre, conçues pour faciliter l'apprentissage du langage et de son support, ainsi que de la possibilité de manipuler les fichiers produits.

### A. Ecran d'accueil.

Comme nous l'avons exposé, la liste des commandes est en fait un menu. Dès lors, et dans la mesure de la place disponible, il nous semble nécessaire d'en faire apparaître tous les éléments à l'écran, éventuellement en deux ou plusieurs colonnes,

en évitant autant que possible d'obliger l'utilisateur à faire défiler son écran. Le recours aux barres de défilement verticale et/ou horizontale devrait donc être limité au seul cas où l'espace disponible serait insuffisant pour afficher la liste dans sa totalité.

Ainsi, l'écran d'accueil comprenant toutes les commandes du langage pourrait être disposé de la même manière que la liste de fichiers dans le MS-DOS Executive: une liste sur plusieurs colonnes avec, en cas de besoin seulement, une barre de défilement horizontale ou verticale. Vu la longueur de certains noms de commande, ceux-ci devraient être affichés en deux colonnes. Cette solution offre l'avantage de pouvoir réorganiser facilement la liste en cas de changement de la taille de la fenêtre de l'application.

Comme nous le soulignons ci-dessous pour les paramètres, l'ordre dans lequel les commandes sont affichées n'est pas quelconque. Dans l'état actuel de l'interface, elles sont présentées par ordre d'utilisation. Cependant, étant donné le nombre important des commandes disponibles, il n'est sans doute pas très aisé de repérer au premier coup d'oeil la commande souhaitée, particulièrement pour un utilisateur peu familiarisé avec le langage de commande. C'est pourquoi nous avons envisagé une présentation alternative, où les commandes sont groupées selon leur fonctionnalité.

Une deuxième solution consisterait donc à incorporer dans la barre de menu les commandes réparties en plusieurs classes constituant autant de menus déroulants ("popup menus"). La taille de la barre de menu ne devrait pas poser de problème, puisque sous Windows elle est automatiquement arrangée en fonction de la largeur de la fenêtre où elle apparaît.

Un avantage de cette présentation est que les commandes ne peuvent apparaître que par classes, guidant ainsi l'utilisateur novice dans son choix. Nous avons cependant retenu une objection majeure: le fait de présenter sur la même barre de menu les commandes propres à l'interface et celles du langage de DSL-SPEC. Dans pareille hypothèse, et en tenant compte des modifications proposées ci-dessous pour les écrans des commandes, la zone cliente de l'application ne serait plus utilisée du tout.

Nous pourrions aussi envisager une solution intermédiaire, dans laquelle l'ensemble des commandes serait affiché dans la zone cliente de l'application sous la forme d'une liste subdivisée en classes.

Avec cette troisième proposition, toutes les commandes apparaissent sur l'écran (pour autant que la taille de la fenêtre de l'application le permette), mais elles sont regroupées selon leur fonctionnalité. Une telle liste serait cependant plus difficile à réorganiser si la fenêtre était redimensionnée.

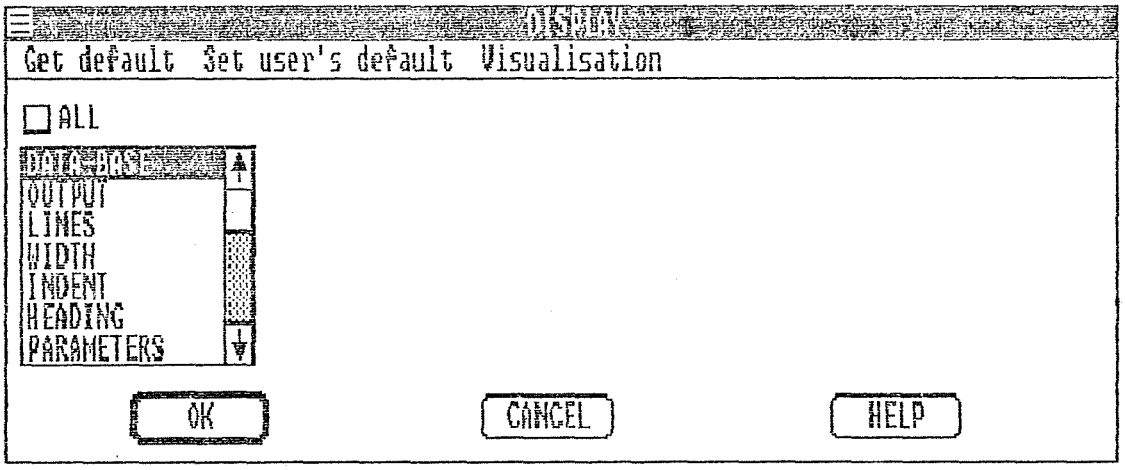
## B. Ecrans des commandes.

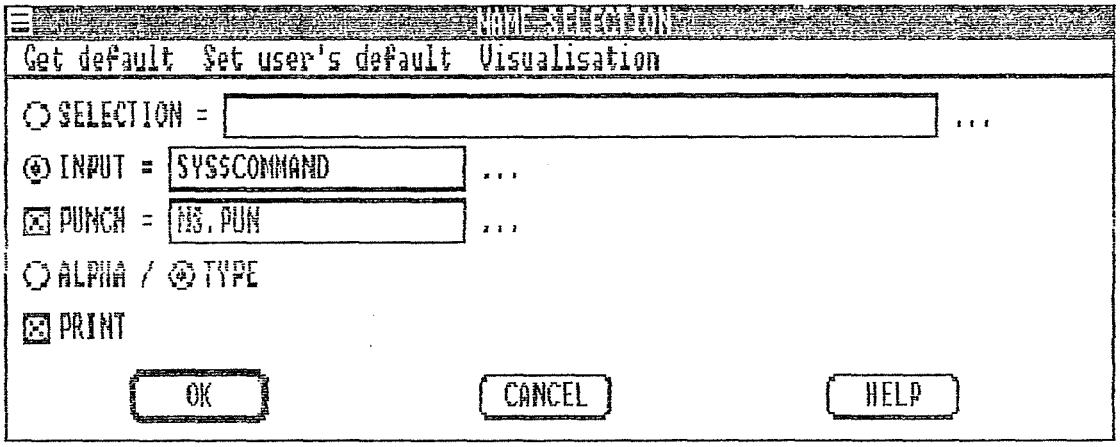
### a. Présentation générale.

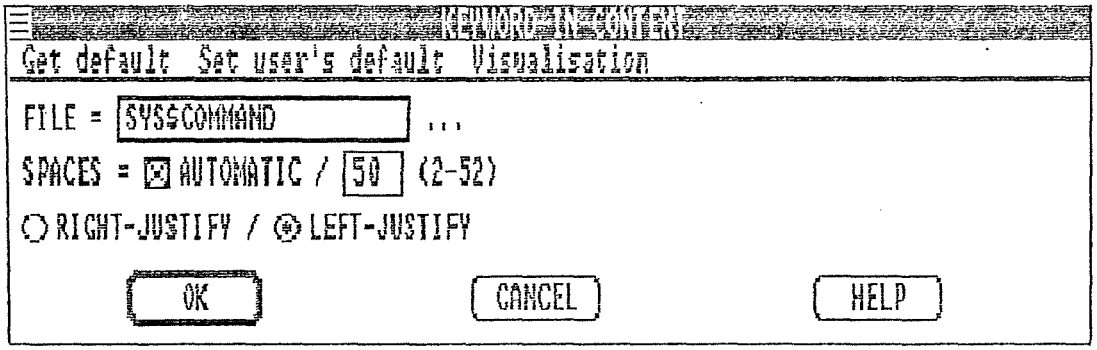
La disposition des paramètres dans les écrans des commandes pourrait se faire à l'intérieur d'une boîte de dialogue utilisant du texte pour le nom des paramètres, des fenêtres d'édition pour l'introduction de valeurs, des "checkboxes" montrant si un paramètre est positionné ou non, des boutons radio pour indiquer une exclusion entre paramètres, et des "listboxes" permettant d'effectuer un choix parmi plusieurs valeurs. Des boutons poussoirs constitueraient un moyen d'accéder à certaines fonctions spéciales: Ok pour valider la commande, Cancel pour l'annuler et Help pour obtenir un complément d'information au sujet de la commande ou de ses paramètres (voir exemples page suivante).

1. Barre de menu.

Il pourrait également être fait usage d'une barre de menu comprenant des commandes telles que attribuer la valeur par défaut de l'utilisateur ou du système, et ce soit pour un seul paramètre (le dernier auquel l'utilisateur a accédé), soit pour tous, ou mémoriser les valeurs par défaut données par l'utilisateur. Ces fonctions disparaîtraient de la barre de menu de l'écran d'accueil. Il nous paraît en effet préférable de ne pas mélanger les fonctions propres à l'édition des commandes avec celles, plus générales, qui font partie de l'écran d'accueil. La fonction d'aide que nous avons d'abord présentée sous la forme d'un bouton


Get default Set user's default Visualisation
ALL
OUTPUT
LINES
WIDTH
INDENT
HEADING
PARAMETERS
OK
CANCEL
HELP


Get default Set user's default Visualisation
SELECTION =
INPUT = SYS$COMMAND
PUNCH = NS.PUN
ALPHA / TYPE
PRINT
OK
CANCEL
HELP


Get default Set user's default Visualisation
FILE = SYS$COMMAND
SPACES = AUTOMATIC / 50 (2-52)
RIGHT-JUSTIFY / LEFT-JUSTIFY
OK
CANCEL
HELP

poussoir pourrait tout aussi bien figurer dans cette barre de menu.

2. Disposition des paramètres.

Dans de telles boîtes de dialogue, il serait possible de grouper deux ou trois paramètres similaires sur la même ligne, dans le but d'optimiser l'utilisation de la surface disponible. Ce serait indispensable pour certaines commandes comptant de nombreux paramètres. Nous obtiendrions alors une distribution beaucoup plus uniforme des informations sur l'écran, tout en réduisant considérablement la longueur de ces écrans, pour lesquels le défilement ne serait d'ailleurs plus nécessaire.

Quant à l'ordre d'affichage des paramètres, qui nous a été donné comme une contrainte à respecter, nous le laisserons tel qu'il est déjà implémenté dans la version précédente de l'interface.

3. Contraintes.

Les dimensions des boîtes de dialogue doivent être telles que tous les paramètres de n'importe quelle commande y trouvent place: c'est pourquoi certains paramètres devraient figurer sur la même ligne. D'autre part, la nature même des boîtes de dialogue sous Windows leur assure une place fixe et suffisante, quelle que soit la taille de la fenêtre dans laquelle elles apparaissent. La seule condition à respecter a trait aux dimensions maximales des boîtes de dialogue: elles ne peuvent en aucun cas excéder la

taille de l'écran, ni recouvrir la zone des icones. De plus, il ne paraît pas vraiment souhaitable que les barres de menu et de titre de l'éditeur de commandes soient cachées par l'une ou l'autre de ces boîtes de dialogue.

En respectant ces règles, l'utilisateur aura donc toujours sous les yeux l'ensemble des paramètres à positionner, ce qui n'est pas le cas s'ils ne peuvent être affichés que les uns en dessous des autres. De la sorte, il peut disposer d'un maximum d'informations sans être obligé de faire défiler son écran.

b. Représentation des paramètres.

1. "Checkboxes".

A l'intérieur des boîtes de dialogue correspondant aux différentes commandes, les paramètres optionnellement préfixés par NO dans le langage de commande seraient précédés ici d'une "checkbox". Nous ne pensons pas qu'il soit vraiment utile de faire changer l'énoncé du paramètre selon qu'il est positionné ou non, comme dans l'interface existante, où le préfixe NO apparaît et disparaît en fonction du positionnement de paramètre. Nous aurions par exemple pour NO CROSS-REFERENCE:

☐ CROSS-REFERENCE

2. Fenêtres d'édition.

Le nom de chaque paramètre de type 'paramètre=valeur' serait suivi soit d'une fenêtre d'édition contenant, le cas échéant, la valeur attribuée par défaut et des bornes entre les-

quelles la valeur doit être comprise dans le cas des entiers, soit de la liste des valeurs possibles.

La taille des fenêtres d'édition devrait dépendre de leur contenu, en réponse à la critique formulée plus haut. Elle devrait être égale à la taille maximale de la valeur à introduire dans les cas où cela est possible (entiers, chaînes de caractères, noms d'objets DSL). De plus, l'espace maximal disponible pour l'introduction de ces valeurs serait visible à l'écran, plutôt que de n'en signaler la limite à l'utilisateur que par un signal sonore Dans les autres cas (noms de fichiers et listes d'objets), le défilement de la fenêtre d'édition devrait être automatique, sa longueur serait réduite, mais de manière à ce que le contenu en soit totalement visible dans la plupart des cas.

Exemples:

Entier: LINKS = [ 58]

Chaîne de caractères: BOX-CHARS = [ ]

3. "Listboxes".

La liste des valeurs possibles aurait la forme d'une "listbox", permettant de sélectionner une ou plusieurs valeurs conformément à la syntaxe du langage de commande et ne devrait être munie d'une barre de défilement que si toutes les valeurs ne peuvent pas y apparaître en même temps. Cependant, en vue d'assurer l'extensibilité du système, chaque "listbox" pourrait comprendre une barre de défilement verticale, qui serait cachée automatiquement lorsqu'elle est inutile.

4. Boutons radio.

Dans les cas où il n'y a que deux valeurs possibles ("switches"), il ne paraît pas très approprié d'utiliser une "listbox", mais plutôt d'afficher les deux valeurs précédées chacune d'un bouton radio , indiquant quelle est la valeur actuellement attribuée au paramètre. Cependant, si les deux valeurs alternatives sont ON et OFF, il serait sans doute plus simple de se contenter d'une seule "checkbox" devant le nom du paramètre, comme nous le proposons ci-dessus pour les paramètres de type "(NO)paramètre". Il est en effet permis de se demander pourquoi des paramètres ayant une signification similaire sont différenciés de la sorte dans le langage de commande. Ils auraient l'aspect suivant:

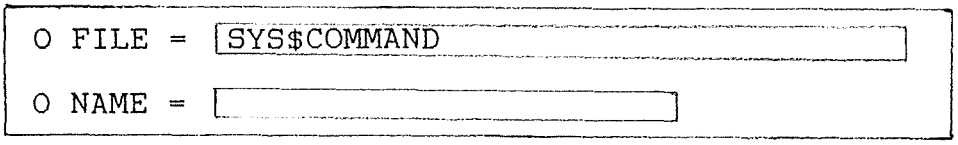
MODE = O BATCH / ⊙ TERMINAL

O RIGHT-JUSTIFY / ⊙ LEFT-JUSTIFY

5. Exclusions.

Les paramètres s'excluant mutuellement seraient précédés chacun d'un bouton radio permettant de choisir soit l'un soit l'autre. Ils pourraient également être groupés dans un cadre ("groupbox") .

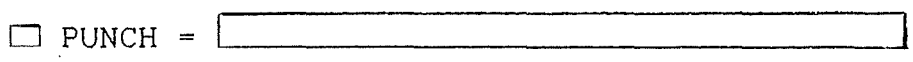
Exemple:



6. Combinaisons.

Quant aux paramètres optionnellement préfixés par NO et auxquels peut également être attribuée une valeur, (comme par exemple l'option PUNCH de plusieurs commandes), il serait sans doute utile de distinguer les différentes formes qu'ils peuvent prendre d'une façon plus claire que dans l'interface actuelle, où ils apparaissent deux fois: d'abord sous la forme "NO paramètre" et puis "paramètre = valeur", ces deux formes étant mutuellement exclusives. Une solution acceptable serait de faire suivre le nom du paramètre des options qu'il peut prendre avec un bouton radio (option NO et fenêtre d'édition avec la valeur par défaut), par exemple:


Une seconde manière de représenter ces paramètres serait de leur adjoindre une "checkbox" indiquant s'ils sont positionnés ou non (comme pour les "switches"), et de placer la fenêtre d'édition derrière leur nom. Cette proposition a l'avantage de la concision à l'écran, tout en maintenant la cohérence avec la représentation proposée pour les autres types de paramètres. Ainsi l'exemple précédent serait représenté par:




Un dernier type de paramètres admet soit une valeur prédéfinie dans le langage, soit une valeur à introduire par l'utilisateur. Plutôt que de distinguer ces deux cas, comme dans l'inter-

face actuelle, nous pensons que la valeur prédéfinie pourrait aisément servir de valeur par défaut au paramètre et être affichée dans la fenêtre d'édition, où une autre valeur peut être introduite. Cette présentation rencontre l'objectif de concision à l'écran, tout en conservant la dualité des valeurs possibles: soit une valeur prédéfinie, qui serait dès lors considérée comme le défaut du système, soit une valeur propre à l'utilisateur. Par exemple:

OBJECTS = [ALL]

7. Fichiers DOS.

Enfin, dans le cas où la valeur à introduire est un nom de fichier DOS (nous pensons ici au fichier batch invoqué dans l'écran d'accueil), il pourrait être fait appel à une boîte de dialogue semblable à celles utilisées dans nombre d'autres applications conçues pour l'environnement Windows: une boîte de dialogue contenant une fenêtre d'édition avec éventuellement une valeur par défaut pour le nom du fichier, une fenêtre de texte contenant le répertoire courant, une "listbox" avec des noms de fichiers, de lecteurs et de sous-répertoires et une paire de boutons poussoirs Open et Cancel.

Une telle boîte de dialogue offre plusieurs avantages par rapport à l'introduction du nom de fichier actuellement implémentée: liste de fichiers dont le nom répond à un critère donné, facilité pour spécifier un chemin d'accès, possibilité de visualiser le contenu des directories, faible taux d'erreurs, ... En

fait, l'utilisateur n'a plus vraiment besoin de mémoriser le nom exact du fichier et l'endroit où il se trouve, puisqu'il peut facilement retrouver ces informations à n'importe quel moment grâce à la boîte de dialogue. Il y a cependant une restriction à leur utilisation: il est en effet impossible de lister des fichiers stockés sur le site central.

8. Remarque.

Les paramètres obligatoires doivent pouvoir être aisément distingués des autres. Nous pensons que le meilleur moyen de le faire est de leur adjoindre un symbole particulier, ou de les souligner.

c. Conclusion.

Une représentation des paramètres telle que nous la proposons, permettant à l'utilisateur de visualiser en une fois les différentes valeurs prédéfinies qu'il peut donner au paramètre, nous paraît de loin plus facile et plus agréable à utiliser que celle précédemment implémentée, où les valeurs possibles ne sont accessibles qu'à tour de rôle en double-cliquant sur le paramètre.

Le but visé par l'utilisation de boîtes de dialogue telles que nous venons de les décrire est de permettre une manipulation directe des paramètres et de leur valeur. Il s'agit également pour l'utilisateur d'avoir une vue globale sur l'ensemble des paramètres, répartis assez uniformément sur l'espace disponible, en veillant cependant à ne pas surcharger les écrans.

Ainsi, il conviendrait de prendre tout particulièrement en compte l'aération dans les boîtes de dialogue, par exemple en y insérant des lignes blanches. C'est également la raison pour laquelle la taille de certaines fenêtres d'édition devrait être limitée, sans toutefois nuire à la lisibilité de leur contenu. La solution retenue pour chaque type particulier de paramètres sera un compromis entre de telles considérations.

## C. Utilisation de la souris et du clavier.

Dans le cas où le défilement de l'écran ou d'un champ d'information s'impose, il serait bien utile de pouvoir l'effectuer aussi bien avec le clavier qu'avec la souris, et de permettre ce défilement également page par page et à l'aide de l'ascenseur de la barre de défilement.

De même, le déplacement dans la liste de commandes ou de paramètres devrait pouvoir se faire également à l'aide du clavier: par exemple, utilisation des touches du curseur (flèches sur le clavier) pour les déplacements horizontaux et verticaux d'une ligne dans la liste des commandes, des touches PageUp et PageDown pour les déplacements page par page, de la touche TAB pour passer d'un paramètre à l'autre, des touches HOME et END pour se positionner respectivement en début et en fin de liste. L'utilisation de ces touches est une pratique courante dans les applications tournant sous Windows.

Toujours dans l'optique de ne pas pénaliser injustement l'utilisateur qui préfèrerait se servir du clavier, il serait indispensable de lui fournir un accès aux différentes commandes des menus. Cela se ferait au moyen d'"accélérateurs", également d'un usage très répandu sous Windows. Par exemple, l'utilisateur devrait taper l'initiale de la commande souhaitée tout en maintenant la touche CONTROL enfoncée.

Nous pensons en effet que, en raison des préférences variant d'une personne à l'autre, une "bonne" interface devrait être utilisable de manière équivalente avec le clavier et la souris.

D. Traitement des erreurs.

En ce qui concerne le contrôle de validité syntaxique des valeurs introduites, il doit être le plus puissant possible, au moment même de la saisie ou au plus tard avant de quitter la boîte de dialogue dans laquelle la saisie s'est opérée. En fait, il y a trois moments où la validation d'une commande ou d'un paramètre peut prendre place: soit lorsque l'utilisateur en a terminé avec l'édition de la commande et qu'il y fait explicitement appel en cliquant sur le bouton poussoir Ok, soit dès qu'il quitte une fenêtre d'édition, soit enfin chaque fois qu'il y introduit un caractère. Il est clair que la première de ces trois possibilités est inacceptable, surtout avec des commandes pouvant compter de

nombreux paramètres. L'utilisateur se retrouve éventuellement avec plusieurs erreurs dont il ne peut être averti qu'en cliquant sur le bouton Ok jusqu'à ce que sa commande soit acceptée. Toutefois, c'est quand même le moment idéal pour vérifier, par exemple, si un paramètre obligatoire a reçu une valeur.

Avec des boîtes de dialogue telles que nous les proposons, la manière la plus simple d'avoir un contrôle syntaxique quasi immédiat est d'y faire appel dès que l'utilisateur sort de la fenêtre d'édition où l'introduction d'une valeur a eu lieu, par exemple quand il clique sur un autre paramètre ou sur Ok. Il est également possible de n'accepter que des caractères valides lors de l'introduction d'une valeur, mais ce type de validation reste cependant incomplet: par exemple, comment, dans ce cas, vérifier si le nombre introduit est effectivement compris entre deux valeurs données?

Quel que soit le moment où ce contrôle de validité syntaxique a lieu, les messages d'erreur doivent être le plus précis et compréhensible possible, et, idéalement, indiquer ce que l'utilisateur doit faire pour remédier à son erreur. Par exemple, plutôt que de se contenter d'indiquer à l'utilisateur qu'il a introduit un caractère invalide, il faudrait lui faire savoir lequel et quels sont les caractères permis. Il serait également utile que les messages d'erreur fassent mention du paramètre où l'erreur a été décelée. Cet aspect est particulièrement important si la validation a lieu à la fin du dialogue. De plus, il faut permettre

à l'utilisateur de corriger facilement son erreur, sans devoir tout réintroduire. En clair, il s'agit de laisser visible la valeur erronnée après le message d'erreur.

Dans le même ordre d'idées, il pourrait s'avérer profitable de disposer d'une fonction permettant de rééditer la commande précédente. Il serait dès lors facile de récupérer une erreur (sémantique ou autre) indétectable par l'interface, et ce sans devoir recommencer tout le processus de définition et de positionnement des paramètres. Cette fonction existe déjà de manière implicite: il suffit de rééditer la commande.

E. Fonction d'aide.

L'utilisateur devrait pouvoir disposer d'une fonction d'aide, couvrant au minimum le format des valeurs à introduire, leur taille, les caractères permis, etc. Une fonction d'aide plus étendue, décrivant brièvement l'usage des différentes commandes et de leurs paramètres, pourrait également s'avérer bénéfique, spécialement pour l'utilisateur débutant: cela lui éviterait souvent de recourir au manuel de référence. Une telle fonction serait accessible, pour le paramètre (ou la commande) sélectionné en dernier lieu, soit par un bouton au bas de l'écran (à côté des boutons Ok et Cancel), soit par une option dans le menu de la boîte de dialogue. Nous pourrions aussi lui consacrer une touche de fonction, comme c'est le cas dans de nombreux programmes.

Dans l'interface actuelle, les informations concernant la valeur à introduire sont reprises dans le texte des boîtes de dialogue: type de valeur, bornes pour les entiers, longueur pour les chaînes de caractères, nombre d'objets pour les listes, ainsi que, au moyen d'un signal sonore, la longueur maximale de la valeur. Ces informations apparaîtront sans aucun doute fort maigres à l'utilisateur qui ne maîtrise pas bien le langage.

## F. Fichiers.

Dans le cas de l'utilisation de DSL-SPEC en mode batch, il serait souhaitable de pouvoir manipuler le fichier utilisé sur la station de travail, et plus seulement sur le site central. Il faut pouvoir modifier ou supprimer une commande déjà envoyée, et insérer une nouvelle commande n'importe où dans le fichier. De même, l'utilisateur devrait pouvoir disposer des fonctions élémentaires de gestion de fichier (ouvrir et sauver). Il faudrait également lui permettre de visualiser le contenu de son fichier de commandes.

Ces différentes facilités étant présentes dans l'éditeur de texte libre, nous pensons que le mieux est de se servir de celui-ci, plutôt que d'en inclure un nouveau dans l'éditeur de commandes de DSL-SPEC.

La possibilité d'envoyer un fichier de commandes à l'émulateur de terminal ou dans un fichier batch permettrait une

utilisation efficace de "macros", nécessaire pour tout utilisateur quelque peu expérimenté dans le maniement de DSL-SPEC. Cependant, étant donné que l'émulateur de terminal n'accepte qu'une seule commande à la fois, l'envoi d'un fichier entier poserait des problèmes de synchronisation entre l'émulateur et l'éditeur de commandes. Il est donc permis de penser qu'une telle facilité ne puisse pas être implémentée du moins de manière efficace pour le mode interactif.

## G. Fenêtres d'état.

L'utilité des deux fenêtres d'état peut être mise en doute. En effet, en se servant de boîtes de dialogue telles que nous les proposons, le nom de la commande en cours de composition peut apparaître dans le titre de la boîte de dialogue correspondante. Il n'est donc pas nécessaire de lui consacrer une fenêtre supplémentaire. Quant au mode de fonctionnement, il pourrait tout aussi bien en être fait mention dans la barre de titre de l'application, de même que, le cas échéant, le nom du fichier de commandes en cours d'édition.

## H. Visualisation des commandes.

Il serait très intéressant d'offrir à l'utilisateur la possibilité de visualiser la commande en cours de composition, telle qu'elle sera envoyée à l'émulateur de terminal ou dans un

fichier. Cette option lui permettrait de se familiariser beaucoup plus rapidement avec le langage de commande de DSL-SPEC et ses nombreuses abréviations. Dans l'état actuel, l'utilisateur ne peut voir sa commande "traduite" dans le langage de DSL-SPEC que dans la fenêtre de l'émulateur de terminal, après qu'elle y ait été envoyée. Il faudrait pour cela une fenêtre supplémentaire de dimensions réduites (par exemple deux lignes) et munie d'une barre de défilement verticale.

Si cette fenêtre demeurait visible pendant que l'utilisateur poursuit l'édition de sa commande, elle devrait être mise à jour lors de toute modification des paramètres.

En cas d'utilisation d'un fichier de commandes, la visualisation de la ligne de commande pourrait se faire directement dans la fenêtre d'édition du fichier. Nous pensons néanmoins qu'il serait préférable d'utiliser deux fenêtres distinctes, comme en mode interactif, et ce afin d'éviter toute confusion.

## 4. ENQUETE.

Lorsque nous avons présenté les nouvelles spécifications de l'interface Windows pour l'éditeur de commande de DSL-SPEC, que nous venons d'exposer dans la première partie de ce chapitre, à des personnes ayant acquis une certaine expérience du langage de commande et de l'utilisation de la station de travail DSL, nos propositions ont été, dans l'ensemble, accueillies comme une réelle amélioration.

Les échanges de vues que nous avons eus avec ces utilisateurs nous ont amené à préciser certaines idées, à arbitrer des choix entre des propositions alternatives, et à réviser notre solution intuitive de certains aspects du problème. De plus, quelques suggestions nous ont été soumises, qui devraient rendre cette interface non seulement plus ergonomique et facile à utiliser, mais aussi et surtout plus cohérente.

Nous allons maintenant exposer successivement les principales idées qui nous ont été fournies à cette occasion.

### A. Ecran d'accueil.

En ce qui concerne la présentation du menu des commandes, il serait souhaitable que celles-ci apparaissent groupées par classes. De plus, l'utilisation de menus déroulants introduirait une indirection supplémentaire. La solution préférée serait donc d'afficher les commandes en colonnes dans la zone cliente, avec un

titre indiquant la classe de commande. Idéalement, chaque classe de commandes devrait apparaître en entier, mais cela impliquerait sans aucun doute l'implémentation d'une puissante fonction pour gérer l'affichage selon la place disponible. Il ne faut pas non plus oublier l'aération au sein de ce menu (lignes blanches entre les classes, répartition des colonnes, ...).

## B. Ecrans des commandes.

### a. Présentation générale.

Un désavantage de la présentation telle que nous l'avons initialement proposée est que certains paramètres risquent de n'être pas vus par l'utilisateur, qui doit parcourir la liste tantôt verticalement, tantôt horizontalement. Par exemple, il arrive qu'un "switch" soit intercalé entre deux longues fenêtres d'édition. Nous retrouvons en outre le problème de l'affichage des paramètres sur plusieurs colonnes.

La disposition des paramètres pourrait donc être plus homogène. Le critère d'ordonnancement sur lequel nous nous sommes basé est l'importance et la fréquence d'utilisation des paramètres, ce qui entraîne des disparités dans la présentation. Nous pourrions envisager d'autres critères, comme par exemple grouper tous les paramètres similaires. Nous obtiendrions ainsi une présentation "pyramidale": en haut de la boîte de dialogue, les paramètres munis de "checkboxes", puis ceux demandant l'introduction de quelques caractères ou d'une valeur entière, pour finir

avec les paramètres les plus longs, ou, inversement, les afficher du plus long au plus court.

Nous pensons qu'une bonne solution devrait prendre en compte ces différents aspects: s'il est impératif qu'un paramètre obligatoire soit mis en évidence, il conviendrait également de rassembler ceux qui ont la même apparence.

b. Dépendances entre paramètres.

Une seconde suggestion pour les boîtes de dialogue a trait aux dépendances entre paramètres: certains d'entre eux n'ont de signification que si d'autres sont positionnés. Comme exemple de ces dépendances, nous pouvons citer la commande STRUCTURE, où les paramètres ROW-ORDER et COLUMN-ORDER n'ont de sens que si RELATIONS-MATRIX est positionné. Ces paramètres ne seraient affichés que s'ils peuvent être pris en compte, ou du moins ils seraient regroupés dans un cadre commun.

Une variante de cette suggestion serait de mettre les dépendances en évidence par l'utilisation de boîtes de dialogue en deux niveaux: dès que le paramètre à l'origine de la dépendance est positionné, une boîte de dialogue apparaît sur la première avec tous les paramètres dépendants. Une telle solution présente un autre avantage, non négligeable pour certaines commandes: la liste des paramètres affichés dans la boîte de dialogue principale serait raccourcie, et la présentation s'en trouverait fortement allégée. Mais cela introduirait une indirection supplémentaire dans le positionnement des paramètres.

Etant donné les complications que de telles suggestions ne manqueraient pas d'entraîner lors de l'étape d'implémentation, sans oublier le fait que le caractère généralisable de l'interface à d'autres langages risquerait fort d'en souffrir, il a été décidé de ne pas pousser plus avant dans cette voie. Nous reconnaissons cependant volontiers le bien-fondé de ces suggestions, du moins dans leur principe.

c. Fenêtres d'édition.

Pour tous les paramètres ayant une fenêtre d'édition précédée soit d'une "checkbox" soit d'un bouton radio, tout changement dans la fenêtre devrait s'accompagner du positionnement automatique de la "checkbox" ou du bouton. Ce serait le cas notamment pour les paramètres PUNCH, FILE, NAME, etc... figurant dans de nombreuses commandes.

Etant donné la place disponible, l'utilisation de fenêtres défilantes s'impose pour certains paramètres. Il s'agit de fenêtres dont le contenu, éventuellement plus grand que la longueur de la fenêtre, peut défiler ("scroll") automatiquement ou avec les touches du curseur. On peut cependant se demander s'il ne serait pas bien utile de faire savoir à l'utilisateur que la fenêtre est défilante et donc que tout le contenu de cette fenêtre est effectivement pris en compte. Le signaler dans le manuel d'utilisation ne nous paraît pas suffisant. Ce qu'il faudrait, c'est le voir à l'écran. Une solution consisterait à employer des fenêtres de deux

lignes, mais nous restons avec le problème de savoir où scinder la valeur introduite, sans compter l'espace supplémentaire occupé par une telle fenêtre. Finalement, le plus simple nous a semblé de faire suivre la fenêtre d'édition défilante d'un symbole particulier, en l'occurrence des points de suspension.

Une suggestion nous a été faite selon laquelle, dans les boîtes de dialogue, les fenêtres d'édition de certains paramètres n'apparaîtraient que si le paramètre est positionné. Ce serait par exemple le cas pour le paramètre PUNCH. Cela permettrait d'améliorer la lisibilité de certains écrans en diminuant leur charge visuelle. Il faut cependant voir dans quelle mesure une telle suggestion serait réalisable. C'est la raison pour laquelle cette idée n'a pas pu être prise en considération.

d. Paramètres exclusifs.

Le cadre entourant des paramètres exclusifs, comme par exemple les paramètres FILE et NAME de nombreuses commandes, peut induire l'utilisateur en erreur quant à leur importance. Ce problème pourrait être résolu par exemple en limitant le cadre aux seuls boutons radio devant ces paramètres.

Dans le but de faciliter l'implémentation du générateur de dialogue, et devant l'objection que de tels cadres surchargeraient l'écran de manière peu opportune, nous avons décidé de ne pas en tenir compte. Toutefois, ils peuvent toujours être ajoutés aux boîtes de dialogue à l'aide de l'éditeur de dialogues de Windows,

dont nous reparlerons par la suite.

e. Valeurs de types différents.

Certains paramètres peuvent prendre soit une valeur entière, soit une valeur par défaut alphabétique. Plutôt que d'afficher cette dernière dans la fenêtre d'édition, il serait sans aucun doute préférable de distinguer les deux cas de manière plus visible, ce qui pourrait se faire en adoptant la présentation suivante:

SPACES = AUTOMATIC / [ 10 ] (2-100)

Tout changement dans la fenêtre d'édition doit entraîner le dépositionnement de la valeur AUTOMATIC (disparition de la croix dans la "checkbox").

f. Boîtes de dialogue à deux niveaux.

Puisque les relations à spécifier pour le paramètre RELATION figurent dans un fichier, elles pourraient très bien être proposées à l'utilisateur sous forme d'une "listbox" à choix multiple. Celle-ci apparaîtrait dans une boîte de dialogue supplémentaire (avec deux boutons "Ok" et "Cancel"), au-dessus de celle de la commande. Les relations choisies seraient alors concaténées et affichées dans la fenêtre d'édition, en respectant le format prescrit par la syntaxe du langage.

Il en va de même pour les noms de fichiers à attribuer au paramètre STATEMENTS de plusieurs commandes.

Nous pourrions envisager de limiter l'accès à cette fenêtre d'édition, de manière à empêcher l'utilisateur d'y introduire des erreurs. Cependant, un utilisateur quelque peu expérimenté préfèrera sans doute garder la possibilité d'introduire et de modifier directement la valeur introduite à l'intérieur de la fenêtre d'édition, plutôt que de subir un niveau d'indirection supplémentaire.

C. Fichier de commandes.

Les avantages entraînés par la possibilité de modifier directement une commande dans un fichier, notamment une plus grande souplesse dans l'utilisation de DSL-SPEC, doivent être mis en balance avec le risque d'y introduire des erreurs.

# 5. SPECIFICATION DE L'INTERFACE A IMPLEMENTER.

Tout au long de ce chapitre, nous avons décrit de nombreuses possibilités d'améliorer l'interface de l'éditeur de commandes DSL-SPEC. Nous nous proposons à présent de préciser les choix qui ont été posés en vue de l'implémentation de la nouvelle interface. Pour ce faire, nous reprendrons les différents aspects abordés dans les pages précédentes, en spécifiant quelle solution a été retenue et en fonction de quels critères.

## A. Ecran d'accueil.

Comme l'enquête que nous avons menée l'a clairement mis en évidence, et afin de répondre au manque de structure dans la liste des commandes, celle-ci sera désormais hiérarchisée en plusieurs classes regroupant les commandes selon leur fonctionnalité.

Le nom des différentes classes figurera dans un menu déroulant 'Command classes' de la barre de menu associée à l'écran d'accueil, en-dessous de l'option 'All'. De la sorte, l'utilisateur pourra, selon son choix, avoir une vue globale ou partielle des commandes du langage. La décision de proposer ces classes dans un menu déroulant renforce encore la distinction entre les commandes du langage et celles propres à l'interface.

D'autre part, la barre de menu de l'écran d'accueil sera expurgée de toutes les commandes qui n'y sont pas nécessaires, en réponse à l'objection que nous avions formulée lors de la critique

de l'interface, à savoir le mélange des commandes se rapportant aux deux types d'écrans. La barre de menu ne contiendra donc plus que les commandes relatives au choix du mode de fonctionnement de l'éditeur et à l'affichage de la liste des commandes du langage.

Les commandes seront affichée en plusieurs colonnes et, comme nous l'avions suggéré, une barre de défilement horizontale ne sera présente que si elle est nécessaire. De plus, il sera permis de faire défiler l'écran colonne par colonne ou page par page et ce aussi bien avec la souris que par les touches du curseur sur le clavier. L'affichage du premier ou du dernier écran ou à partir d'un point précis sera également possible.

Cette manière de présenter la liste des commandes apporte une solution tout à fait réalisable aux critiques que la précédente version de l'interface avait soulevée à ce sujet (voir Figure 4).

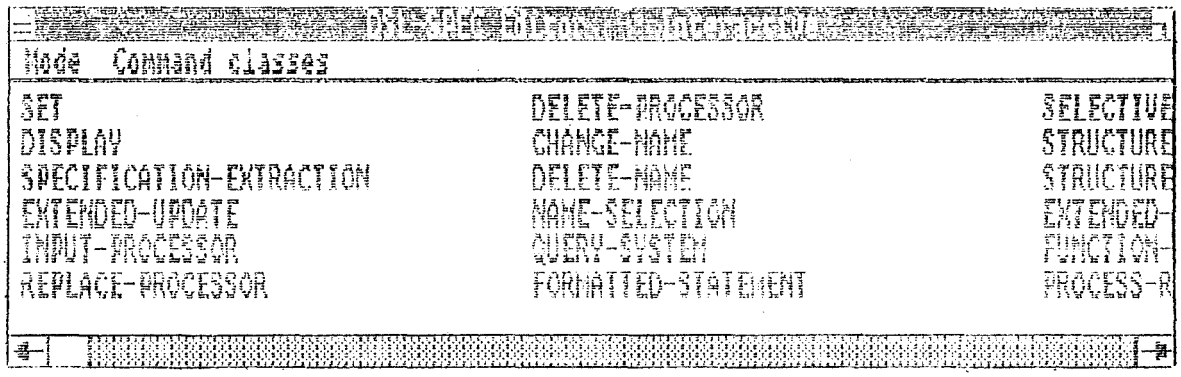



Figure 4.

## B. Ecrans des commandes.

### a. Présentation générale.

En ce qui concerne la présentation générale de ces écrans, nous nous en tiendrons aux propositions que nous avons initialement avancées, sans toutefois placer deux ou plusieurs paramètres de manière automatique sur la même ligne.

Les changements éventuels dans l'ordre ou la disposition des paramètres pourront toujours être pris en charge par l'éditeur de dialogues.

Afin de supprimer une étape qui n'était pas vraiment utile dans la définition des commandes, leur envoi dans le fichier batch ou à l'émulateur du terminal sera directement associé au bouton poussoir Ok des boîtes de dialogue.

D'autre part, étant donné la difficulté de définir des écrans liés à une fonction d'aide vraiment efficace, cet aspect sera provisoirement laissé en suspens.

La barre de menu de ces boîtes de dialogue contiendra les options nécessaires à la mémorisation des valeurs par défaut de l'utilisateur et à l'initialisation d'un seul ou de tous les contrôles avec ces valeurs ou celles du système, ainsi que l'accès à la fenêtre de visualisation. Les fonctions liées à chacune de ces options devront être implémentées.

b. Les contrôles.

En guise de spécification pour cette partie essentielle de l'interface, nous allons reprendre de manière plus formelle les différentes propositions de présentation des paramètres dans les boîtes de dialogue des commandes telles qu'elles ont été adoptées: c'est ce que nous appellerons les "types visuels". Dans les cas ou plusieurs alternatives étaient possibles, notre choix a été déterminé par des critères de cohérence par rapport aux autres types visuels et de concision dans la présentation, de manière à garder la densité globale d'informations à l'écran à un niveau acceptable dans toutes les boîtes de dialogue. Les types visuels sont des combinaisons des éléments de base suivants que nous désignerons sous le terme générique de 'contrôles': "checkbox", fenêtre d'édition ("editbox"), "listbox", bouton radio et enfin du texte, servant à compléter les autres éléments de base, principalement les fenêtres d'édition et les "listboxes". Les séparateurs ("=" et "/") peuvent être soit affichés séparément, soit incorporés au label du contrôle qui les précède. C'est cette dernière solution que nous avons choisie, pour des raisons de facilité, essentiellement dans l'utilisation de l'éditeur de dialogues.

D'après les propositions que nous avons faites, nous pouvons distinguer les types visuels ci-dessous. Pour chacun d'eux, nous mentionnons le(s) type(s) de contrôles à générer et les interrelations éventuelles entre ces contrôles, ainsi qu'un exem-

ple. Les "checkboxes" et les boutons radio sont libellés au nom du paramètre ou de la valeur qu'ils représentent.

1. "Switches" simples.

Une première catégorie de paramètres booléens sera représentée par des "checkboxes". Il s'agit de tous les paramètres optionnellement précédés de NO dans la syntaxe du langage, de même que les paramètres prenant une des valeurs prédéfinies ON ou OFF.

Exemple:

☐PRINT

2. "Switches" doubles.

Pour les autres paramètres booléens, le mode de représentation adoptée utilisera, comme nous l'avons indiqué dans le chapitre précédent, des boutons radio. Ce sont soit des paramètres mutuellement exclusifs, soit des paramètres pouvant prendre deux valeurs prédéfinies mutuellement exclusives. Pour ces derniers, le nom du paramètre apparaît dans un contrôle texte.

Exemples:

⊙ RIGHT-JUSTIFY / O LEFT-JUSTIFY

MODE = O BATCH / ⊙ TERMINAL

3. "Listboxes" simples.

Les listes de plus de deux valeurs prédéfinies pouvant être assignées à un même paramètre apparaîtront sous la forme de "listboxes", le nom du paramètre étant contenu dans un contrôle texte. Puisque, dans la syntaxe du langage, ces différentes

valeurs s'excluent l'une l'autre, la "listbox" sera définie comme n'acceptant pas de choix multiple. D'autre part, il s'est révélé préférable que ces valeurs apparaissent toutes en même temps sur l'écran. La taille de la "listbox" dépendra donc directement du nombre de valeurs à y afficher. De la sorte, la barre de défilement verticale n'est plus nécessaire et sera donc enlevée.

Exemple:

INPUT-OUTPUT-FLOW = [ALL / MESSAGES / DATA-STRUCTURE]

4. "Listboxes" défilantes.

Au cours de la conception de l'interface, il est apparu que la meilleure manière de présenter une liste de paramètres auxquels aucune autre valeur ne doit être attribuée serait de les inclure dans une "listbox". Ce serait le cas pour les paramètres des commandes SET et DISPLAY. Etant donné le nombre relativement important de ces paramètres, l'emploi d'une barre de défilement verticale s'impose. D'autre part, puisqu'ils peuvent être utilisés conjointement, la "listbox" doit permettre la sélection multiple. Le garnissage des "listboxes" se fera lors de l'initialisation de la boîte de dialogue, avec des valeurs mémorisées dans la structure de données ou dans un fichier séparé pour les boîtes de second niveau.

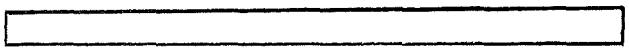
C'est le même type visuel qui sera utilisé pour les boîtes de second niveau des paramètres RELATION et STATEMENTS. Dans ce dernier cas cependant, la sélection doit être unique.

5. Fenêtre d'édition simple.

Pour tous les paramètres acceptant l'introduction d'une valeur par l'utilisateur, une fenêtre d'édition sera utilisée en combinaison avec un ou plusieurs autres contrôles. La fenêtre d'édition, de taille variable selon le type d'information qu'elle doit recevoir, sera défilante si nécessaire, c'est-à-dire pour les noms de fichiers, les listes d'objets DSL et les longues chaînes de caractères. Dans ce cas, rappelons qu'elle sera suivie de points de suspension.

Dans le cas le plus simple, le paramètre sera représenté par un contrôle texte avec son nom et une fenêtre d'édition.

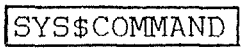
Exemple:

RELATION = 

6. Exclusions.

Si le paramètre fait l'objet d'une exclusion avec un autre, le contrôle texte sera remplacé par un bouton radio. L'introduction d'une valeur dans la fenêtre d'édition devra entraîner le positionnement automatique du bouton radio.

Exemple:

O FILE =  ...

7. Combinaisons.

Quand il s'agit d'une combinaison entre un paramètre booléen et une valeur à introduire, la fenêtre d'édition sera précédée d'une "checkbox", qui devra être positionnée automati-

quement si l'utilisateur fait usage de la fenêtre édition.

Exemple:

☐ PUNCH = [EP.PUN ]

8. Entiers.

Enfin, lorsque la valeur à introduire est un entier, un contrôle texte sera ajouté derrière la fenêtre d'édition, dans lequel seront indiquées les valeurs minimale et maximale acceptables. De plus, si la valeur par défaut de ce paramètre n'est pas entière une checkbox apparaîtra entre le contrôle texte portant le nom du paramètre et la fenêtre d'édition. Dans ce cas, l'introduction d'une valeur impliquera le dépositionnement automatique de la checkbox.

Exemples:

LINKS = [ 50 ] (1-1000)

HORIZONTAL-BOXES = AUTOMATIC / [ 10 ] (2-100)

La définition de ces types visuels nous a paru essentielle, dans la mesure où ils sont à la base du raisonnement suivi dans la construction à la fois du générateur de dialogues et de plusieurs des fonctions principales de l'éditeur de commande.

## C. Fenêtre de visualisation.

La réalisation de cette facilité pourrait se faire de la manière suivante. Le choix de cette option se fait dans la barre de menu des boîtes de dialogue des commandes. La fenêtre apparaît

alors au bas de la boîte, à la place des trois boutons poussoirs. La forme abrégée de la commande en cours d'édition est affichée sur deux lignes dans cette fenêtre toute simple. Si elle excède ces deux lignes, le reste de la commande peut être accédé à l'aide de la barre de défilement verticale.

Il suffit de cliquer n'importe où dans cette fenêtre pour la faire disparaître et continuer le positionnement des paramètres. Dans ce cas, la commande ne pourrait être visualisée que de manière ponctuelle, ce qui supprime le problème de sa mise à jour soulevé précédemment (voir Figure 5).

D. Traitement des erreurs.

La validation syntaxique de toute valeur introduite par l'utilisateur se fera dès l'instant où la fenêtre d'édition qui a reçu cette valeur est désactivée, c'est-à-dire lorsqu'il poursuit son parcours de la liste de paramètres avec la touche TAB ou par un clic de la souris sur un autre contrôle ou sur un bouton poussoir.

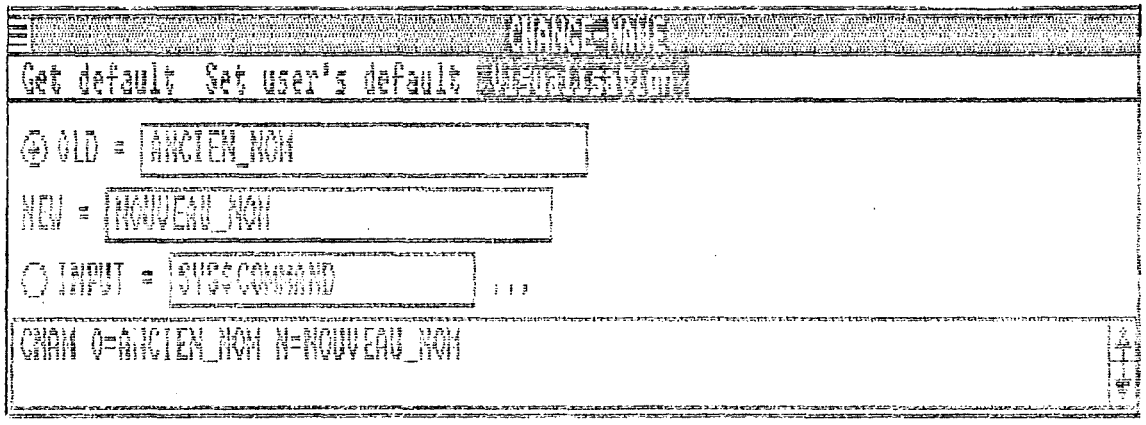



Figure 5.

# CHAPITRE III :

# REALISATION

## 1. INTRODUCTION.

La réalisation des boîtes de dialogue nous amène à nous poser une question fondamentale: devons-nous implémenter une boîte de dialogue générique, valable pour toutes les commandes, ou bien chaque commande doit-elle disposer de sa boîte propre? La solution consistant en une seule boîte de dialogue ne doit pas être retenue pour deux raisons: d'abord, le nombre des paramètres est fort variable selon les commandes, et par conséquent, la surface nécessaire pour certaines commandes serait tout à fait disproportionnée pour d'autres. Ensuite, l'implémentation d'une boîte de dialogue générique serait beaucoup plus compliquée (voire impossible) à réaliser.

De même, la fonction de gestion et de contrôle des dialogues doit-elle être unique ou spécifique à chaque commande? Dans ce dernier cas, la moindre modification dans les paramètres d'une commande exigerait une modification analogue de la fonction associée; il faudrait donc recompiler cette fonction, ce que nous voulons éviter. Il faut donc implémenter une fonction unique, qui puisse permettre l'utilisation éventuelle de l'interface avec un langage de commande différent, et qui soit valable quel que soit le nombre de boîtes de dialogue.

Le générateur de dialogues doit donc produire d'une part les boîtes de dialogue sous une forme reconnaissable par Windows,

c'est-à-dire un fichier de ressources (.RC), et d'autre part un "ensemble de descriptions" utilisables en toute généralité par la fonction de gestion des écrans. Ces descriptions devront contenir toutes les informations nécessaires au dialogue, à son initialisation (valeurs par défaut) et à sa clôture (contrôle de validité, abréviations à générer). Ces descriptions constitueront le contenu du fichier de données.

Dès le départ, il convient de faire la distinction entre l'écran d'accueil (Command Language Interface) et les boîtes de dialogue des commandes, et ce tant pour la représentation visuelle que pour la fonction de contrôle. Cette distinction est issue du mode de représentation de l'interface que nous avons proposé: l'écran d'accueil est un menu, tandis que les boîtes de dialogue des commandes sont des formulaires.

## 2. STRUCTURES DE DONNEES.

### A. Description syntaxique du langage.

La description syntaxique du langage de commande est, comme nous le verrons dans la suite de ce mémoire, absolument primordiale tant pour l'implémentation que pour l'utilisation de l'éditeur de commandes. Nous allons l'aborder en deux parties: d'abord telle qu'elle existait auparavant, puis en décrivant les modifications à y apporter pour qu'elle puisse servir au générateur de dialogues.

#### a. Description originale.

Elle est constituée de plusieurs parties, au sein d'un même fichier, que nous allons à présent passer en revue. Tout d'abord, la première ligne du fichier contient une information plutôt d'ordre technique: le caractère utilisé dans la suite pour délimiter les valeurs par défaut des divers paramètres. La deuxième portion de ce fichier introduit le Command Language Interface. C'est elle qui fournit la liste des commandes du langage ainsi que leur abréviation et leur identifiant. La dernière partie de la description syntaxique du langage est de loin la plus volumineuse. Elle comporte en effet la totalité des commandes et des paramètres que le générateur et l'éditeur auront à manipuler.

Examinons maintenant d'un peu plus près les informations fournies pour chaque commande. La première d'entre elles est son abréviation. Si la commande comporte des paramètres, ceux-ci figurent sur les lignes suivantes, et sont éventuellement accompagnés d'informations supplémentaires. La liste des paramètres se termine par une ligne contenant '999' que nous dénommerons dans la suite "carte 999". Ci-après, nous détaillerons quelque peu, dans l'ordre de leur apparition dans le fichier de descriptions, le genre d'informations susceptibles d'entourer un ou plusieurs paramètres. Chacune de ces informations se trouve sur une ligne séparée commençant par '***WKST***'. Elles indiquent le caractère obligatoire du paramètre, sa valeur par défaut et le ou les autres paramètres qui ne peuvent être utilisés simultanément, ce que nous désignons par le terme "exclusion".

Exemples:

```
***WKST*** Mandatory.
***WKST*** Default is @FILE=SYS$COMMAND@.
***WKST*** Exclude [1-2].
```

Un dernier type d'information figure dans le fichier de description du langage de commande. Elle sert à indiquer une liste de valeurs alternatives prédéfinies que peut prendre le paramètre. Puisque nous les représenterons désormais par des "listboxes", cette information n'est plus nécessaire et sera simplement ignorée par le générateur de dialogues. Il s'agit, par exemple, de:

```
***WKST*** Regroup 5 following lines with previous one.
```

Il est temps à présent de se pencher sur la syntaxe des différents paramètres. Dans les lignes suivantes, les parenthèses désignent des éléments facultatifs, tandis que des éléments alternatifs sont séparés par le caractère "/".

La ligne du fichier qui décrit un paramètre contient un certain nombre d'informations relatives à celui-ci, à savoir son nom, son identifiant et son type. A cela viennent s'ajouter, le cas échéant, son abréviation, sa valeur et l'abréviation de sa valeur. Chaque ligne a la forme générique suivante:

id(*/ (NO))nom(,(N)abr)(=valeur(,abrvaleur)/type)

où: id est l'identifiant du paramètre,

nom est son nom,

abr son abréviation,

abrvaleur l'abréviation de sa valeur,

type est un caractère indiquant le type de valeur que l'utilisateur peut donner au paramètre, c'est-à-dire un nom de fichier (*), un nom d'objet DSL (?), un entier (#), une chaîne de caractères (') ou une liste d'objets DSL (,).

Dans les cas où l'utilisateur peut introduire la valeur que le paramètre prendra, sa description peut être suivie d'une ou deux lignes relatives à cette valeur. Ces lignes commencent par un des nombres 995, 997 ou 998, et nous les dénommerons respectivement "carte 995", "carte 997" et "carte 998". La carte 995 indique le nombre maximal de noms d'objets pouvant figurer dans une liste.

La carte 997 contient le nombre maximal de caractères à introduire: elle est utilisée aussi bien pour les listes d'objets DSL que pour les chaînes de caractères. Enfin, la carte 998 comprend les valeurs minimale et maximale entre lesquelles une valeur entière doit être comprise.

De ces quelques précisions, nous pouvons dès à présent répertorier les paramètres selon les huit modèles suivants, qui serviront de base à l'analyse que notre programme de génération fait du fichier de descriptions:

```
id*nom(,abr)
id NOnom(,Nabr)
id nom(,abr)=*
id nom(,abr)=?
id nom(,abr)=#      suivi de la carte 998
id nom(,abr)='      suivi de la carte 997
id nom(,abr)=,      suivi des cartes 995 et 997
id nom(,abr)(=valeur(,abrvaleur))
```

Les paramètres du premier modèle sont optionnellement précédés, dans le langage, du préfixe 'NO'.

Un identifiant négatif indique un paramètre, ou une valeur, alternatif au précédent. Cette règle n'est d'application que dans le dernier modèle.

Comme nous le verrons lors de l'étude des structures de données nécessaires à l'implémentation de l'interface que nous

avons conçue. les informations contenues dans ce fichier sont indispensables. même si elles doivent être complétées afin que cette interface puisse mieux tirer profit des possibilités de présentation offertes par Windows.

b. Description enrichie.

Comme nous venons de le dire. la description syntaxique originale du langage ne fournit pas à elle seule toutes les informations indispensables à la génération automatique des boîtes de dialogue que nous avons décrites, bien qu'elle en demeure la base incontestable. Plusieurs des décisions de conception que nous avons prises nécessitent un certain nombre d'informations supplémentaires, qui ne pourraient être déduites que très difficilement de la description du langage. C'est la raison pour laquelle il a été décidé d'enrichir cette description. mais sans la transformer véritablement.

Nous allons maintenant passer en revue les différents types d'informations qu'il convient d'y ajouter.

Tout d'abord, puisque la décision a été prise de subdiviser le Command Language Interface en classes de commandes, il faut que le nom de chaque classe y figure et soit aisément discernable des autres descriptions. Afin de préserver l'ordre dans lequel les commandes apparaissent, nous nous proposons d'adopter les deux règles suivantes: tout nom de classe apparaît sur une ligne séparée commençant par '***WKST***'; toutes les commandes apparaissant sous un nom de classe appartiennent à cette classe.

Ainsi, cette information supplémentaire ne doit apparaître qu'entre deux commandes désignées comme appartenant à des classes différentes, et, en outre, il n'est nul besoin de modifier l'ordre des commandes.

Exemple:

***WKST*** Control.

En ce qui concerne la description des paramètres, il s'agit essentiellement d'inclure les informations relatives aux décisions de conception.

Lorsqu'il a été décidé de combiner deux descriptions de paramètres en un seul type visuel, il faut en faire mention par l'information suivante:

***WKST*** Combine 2 following parameters.

D'autre part, la représentation choisie pour certains paramètres nous paraît assez difficile à deviner d'après le seul contexte où ils se trouvent. C'est le cas pour certaines "check-boxes" et "listboxes". De même, il a été décidé de marquer des paramètres mutuellement exclusifs en leur assignant une forme particulière: un bouton radio. Ces différentes informations se retrouveront donc dans la description du langage.

Exemple:

***WKST*** Checkbox.

***WKST*** Radiobutton.

Il convient ici de donner quelques précisions relatives aux paramètres devant figurer dans une "listbox". Nous avons décidé de les entourer de délimiteurs: une ligne spécifiant qu'il s'agit de créer une "listbox" et qui indique le nombre de lignes souhaitées dans cette "listbox", avec optionnellement le mot 'multiple' signifiant qu'un choix multiple doit être permis pour ce paramètre. Le second délimiteur indique la fin de la "listbox". Exemple:

```
***WKST*** Listbox 7 multiple.
***WKST*** End listbox.
```

B. Ressources.

Destiné à être fourni au compilateur de ressources RC de la boîte à outils de MS-Windows, ce fichier doit donc faire référence (par inclusion) au fichier de définitions WINDOWS.H. Deux autres fichiers doivent également y être référencés: il s'agit des fichiers de définitions et de ressources de l'éditeur de commandes. La raison en est très simple: un seul fichier de ressources compilées peut être inclus au programme.

Toutes les informations composant une boîte de dialogue doivent être définies dans un ordre précis et avec des mots-clés déterminés, que nous présentons ci-dessous en majuscules. La forme générique des boîtes de dialogue telles qu'elles ont été définies lors de la conception de l'interface est décrite par les lignes suivantes:

```
id DIALOG posh, posv, largeur, hauteur
STYLE style
CAPTION "nom"
BEGIN
contrôles
DEFPUSHBUTTON "Ok", IDOK, posh, posv, largeur, hauteur, style
PUSHBUTTON "Cancel", IDCANCEL, posh, posv, largeur, hauteur, style
PUSHBUTTON "Help", id, posh, posv, largeur, hauteur, style
END
```

où: id est l'identifiant de la boite de dialogue (du contrôle),

posh et posv les coordonnées de l'endroit où elle (il) prend place,

largeur et hauteur sa largeur et sa hauteur,

style son style,

nom le titre de la boite de dialogue,

contrôles est une suite d'une ou plusieurs des lignes suivantes, dans lesquelles nom est le label du contrôle:

```
LTEXT "nom", id, posh, posv, largeur, hauteur, style
CHECKBOX "nom", id, posh, posv, largeur, hauteur, style
RADIOBUTTON "nom", id, posh, posv, largeur, hauteur, style
EDITTEXT id, posh, posv, largeur, hauteur, style
LISTBOX id, posh, posv, largeur, hauteur, style
```

Nous avons défini le style des boîtes de dialogue de manière à ce qu'elles présentent une barre de titre et qu'il soit possible de leur adjoindre une barre de menu. En ce qui concerne

les contrôles, les styles sont définis en respectant les principes suivant: tous doivent pouvoir être atteints successivement à l'aide de la touche TAB, certaines fenêtres d'édition doivent être défilantes, les valeurs entières doivent être justifiées à droite, les "listboxes" pour lesquelles cela a été spécifié doivent permettre une sélection multiple.

## C. Menu

En ce qui concerne le menu proposé dans l'écran d'accueil de l'éditeur de commandes, les données nécessaires sont, pour chaque commande, son nom (qui sera affiché) et un identifiant permettant de faire la liaison avec la boîte de dialogue correspondante. L'abréviation de la commande est également nécessaire, car elle sera utilisée dans le cas où l'utilisateur souhaite introduire directement une commande, sans passer par la phase de définition et de positionnement des paramètres. La structure de données doit enfin comporter le nom de la classe à laquelle appartient chaque commande.

## D. Données

### a. Quelles données prendre en compte?

Dans les boîtes de dialogue, chaque paramètre est caractérisé par plusieurs données de nature différente: un identifiant (qui fera le lien entre la représentation visuelle et la structure de données), le nom à afficher, les valeurs prédéfinies que le

paramètre peut prendre, son type visuel, sa position à l'intérieur de la boîte de dialogue, son "type logique" (qui intervient dans l'étape de validation syntaxique), des données relatives à ce type logique (bornes pour les entiers, nombre d'éléments et taille des listes, taille des chaînes de caractères, c'est-à-dire, en toute généralité, les données fournies par les cartes 995, 997 et 998 du fichier de descriptions), la valeur par défaut, l'abréviation à générer, ainsi que des indications complémentaires quant au caractère obligatoire du paramètre ou aux relations entre paramètres (exclusions et dépendances).

Ces données auront trait soit à la représentation visuelle du langage de commande (type visuel, position), et ne doivent donc pas être reprises dans la structure de données sur laquelle repose le programme de gestion de l'interface, soit au contrôle des boîtes de dialogue (type logique, valeur par défaut, abréviation), soit aux deux (identifiant). Nous n'expliciterons ici que les données devant figurer dans le fichier de données; les autres ne servent qu'à la composition des boîtes de dialogue et sont des variables du programme de génération.

Le type logique, les données qui lui sont relatives et l'abréviation du parametre interviennent lors de la composition des boîtes de dialogue, dans leur fonction de contrôle et pour faire appel au module de validation syntaxique.

La valeur par défaut est prise en compte lors de l'initialisation de la boîte de dialogue et quand l'utilisateur y fait

explicitement référence en sélectionnant une des options du sous-menu Get Default.

Les exclusions sont prises en compte à la fois dans la composition des boîtes de dialogue (boutons radio), et dans leur fonction de contrôle.

Les dépendances interviennent pour le (dé)positionnement des "checkboxes" et boutons radio lors de l'introduction d'une valeur dans certaines fenêtres d'édition.

b. La structure des données.

Après avoir passé en revue les différentes informations caractérisant les paramètres, nous allons à présent tenter de leur donner une structure qui soit facilement utilisable par l'éditeur de commandes, tout en reflétant autant que possible le mode de représentation choisi.

Dans la structure de données à mémoriser, chaque commande sera donc caractérisée par les informations suivantes: son identifiant, son abréviation, quatre tableaux contenant respectivement la forme abrégée des paramètres et de leurs valeurs prédéfinies, les valeurs par défaut, les chaînes de caractères devant apparaître dans une "listbox" et la forme abrégée de ces dernières, et enfin un ensemble d'"objets" représentant les paramètres.

Que sont ces "objets"? Dans la représentation visuelle choisie, plusieurs contrôles peuvent correspondre à un seul paramètre, comme dans l'exemple suivant:

De plus, à chacun de ces contrôles correspondent des "données" qui lui sont propres à savoir un identifiant, une abréviation et une valeur par défaut éventuelles, ainsi que des messages envoyés par Windows. C'est pourquoi nous avons jugé plus adéquat de choisir ces contrôles comme unité de traitement. Cette solution offre un avantage supplémentaire: il apparaît en effet que toutes les exclusions s'exprimeront désormais uniquement entre deux contrôles, et qu'elles peuvent être généralisées aux cas des paramètres admettant deux valeurs alternatives prédéfinies, comme dans l'exemple ci-dessus, ou une valeur prédéfinie et une autre à introduire.

La structure de données représentant ces contrôles devra donc contenir leur identifiant, une référence à la forme abrégée et à la valeur par défaut, leur type, l'identifiant du contrôle éventuel à exclure, l'identifiant du premier contrôle composant le type visuel et un champ de contrôle servant à marquer un paramètre obligatoire. Nous y ajouterons deux valeurs entières dont le contenu et la signification dépendent du type de contrôle. Ces valeurs serviront à mémoriser les informations fournies par les cartes 995, 997 et 998 du fichier de description du langage, ainsi que le nombre d'éléments à inclure dans les "listboxes" et leur référence.

c. Les données "inutiles".

Quelques précisions s'imposent quant aux informations qui ne se retrouvent pas dans les structures de données. Tout d'abord, en raison de leur longueur, les listes de relations et de fichiers utilisées respectivement avec les paramètres RELATIONS et STATEMENTS sont fournies dans un fichier séparé, et ne font en aucun cas partie de la structure de données que nous venons de décrire. D'autre part, le nom complet des commandes figurera dans la barre de titre des boîtes de dialogue et ne doit donc plus être repris dans la structure de données. Il faut également remarquer qu'il n'est pas besoin de structures de données destinées à mémoriser les valeurs introduites par l'utilisateur. Ces valeurs sont en effet accessibles à tout moment dans la fenêtre d'édition où elles ont été introduites.

E. Valeurs par défaut de l'utilisateur.

La structure de données que nous avons choisie pour supporter cet aspect de l'interface est fort simple et directement inspirée de celle employée pour mémoriser les données relatives aux contrôles. Il s'agit simplement de deux tableaux: l'un contient les valeurs par défaut des contrôles et l'autre les références à ces valeurs. Nous y avons ajouté l'identifiant de la boîte de dialogue correspondante.

## 3. PROGRAMMES.

### A. Processus de génération des boîtes de dialogue.

Après avoir détaillé les données traitées par le générateur de dialogues et l'éditeur de commandes, nous allons à présent exposer comment ces données interviennent dans l'un et l'autre programmes.

#### a. Génération et compilation.

Comme nous l'avons vu précédemment, les boîtes de dialogue produites par le générateur ne peuvent pas être utilisées comme telles par l'éditeur de commandes. Elles doivent d'abord être compilées par le compilateur de ressources RC, ce qui se fait en deux étapes. La première transforme les différentes ressources, dont les boîtes de dialogue générées, de l'éditeur de commandes en un fichier binaire intermédiaire (fichier .RES). Elle est invoquée par la commande DOS:

rc -r file

où file est le nom du fichier contenant les boîtes de dialogue. Celui-ci doit obligatoirement avoir l'extension ".RC", requise par le compilateur de ressources. Le fichier intermédiaire produit portera le même nom, mais avec l'extension ".RES".

Remarquons au passage que cette première étape de compilation aurait pu être intégrée au générateur de dialogues. Nous n'avons cependant pas jugé utile de le faire, afin de ne pas

mélanger deux choses différentes.

b. L'éditeur de dialogue.

Une fois les ressources compilées, les boîtes de dialogue peuvent être modifiées à loisir grâce à l'éditeur de dialogues DIALOG, également compris dans la boîte à outils de MS-Windows. C'est ici que nous pourrons éventuellement grouper plusieurs contrôles dans un même cadre ("groupbox"), et surtout vérifier la hauteur des boîtes de dialogue. Et, le cas échéant, réorganiser la disposition des contrôles. En aucun cas, les boîtes de dialogue ne devraient être plus hautes que la zone cliente d'une application montrée en taille normale lorsqu'elle est la seule à ne pas être iconique.

La modification des boîtes de dialogue est une étape importante dans le processus de génération, car c'est elle, en grande partie, qui sera responsable d'une répartition harmonieuse et suffisamment aérée des contrôles dans les boîtes de dialogue. Lorsqu'un ou plusieurs contrôles ont été déplacés, il faut également veiller à modifier leur ordre, de telle sorte que les déplacements d'un paramètre à l'autre avec la touche TAB n'en soient pas perturbés.

L'éditeur de dialogue peut donc jouer un rôle de tout premier plan dans la réalisation d'une interface telle que nous la proposons. Nous venons de dire qu'il permettait de modifier l'ordre et la répartition des contrôles au sein des boîtes de

dialogue. En poussant le raisonnement jusqu'au bout, nous en arrivons à la conclusion que cet outil pourrait tout aussi bien servir à personnaliser l'interface, par une disposition des contrôles "sur mesure", en fonction de critèrres tels que la fréquence d'utilisation des paramètres ou les préférences spécifiques à certains (groupes d') utilisateurs. Ce serait là un moyen d'atteindre plusieurs des objectifs principaux d'une interface conviviale, notamment une grande satisfaction subjective et des performances accrues dans le chef des utilisateurs concernés.

c. Inclusion de l'interface dans l'éditeur de commandes.

Lorsque celles-ci sont jugées acceptables, la seconde partie de la compilation peut prendre place. Il s'agit en fait d'inclure les ressources compilées, et éventuellement modifiées, dans le fichier exécutable de l'éditeur de commandes. Cela se fait au moyen de la commande DOS:

rc file SPECEDT

où file est le nom du fichier contenant les boîtes de dialogue. Ce fichier doit obligatoirement avoir l'extension ".RES", requise par le compilateur de ressources. SPECEDT est le nom de l'éditeur de commandes, qui doit évidemment se trouver soit dans le chemin d'accès défini par la commande DOS 'PATH', soit dans le répertoire actuel.

Le processus de génération que nous venons de décrire devra être reproduit dans sa totalité chaque fois qu'une modifica-

tion est apportée au langage de commande, ou si nous souhaitons utiliser l'éditeur de commandes avec un langage différent. Il est toutefois possible de récupérer des boîtes de dialogue, modifiées ou non, d'une version à l'autre de l'interface, ce qui se révèle tout à fait indispensable en raison du temps nécessaire à la modification de ces boîtes de dialogue.

## B. Le générateur de dialogues.

### a. Description.

Le programme de génération de dialogues GEN produit, à partir de la description syntaxique d'un langage de commande, des fichiers permettant de présenter ce langage dans l'environnement MS-Windows sous forme de formulaires dans des boîtes de dialogue.

Ces fichiers, dont nous venons de décrire le contenu, sont au nombre de trois: un de ressources, un de données et un de menu. Le premier, écrit en mode texte, contient les définitions des boîtes de dialogue. Il est destiné au compilateur de ressources RC pour être compilé en deux étapes, comme nous l'avons exposé ci-dessus.

Quant aux deux fichiers de données et de menu, GEN les baptise respectivement 'SPECEDT.DTA' et 'SPECEDT.MNU'. Ils ne devraient pas être renommés, car le programme de gestion de l'interface, SPECEDT, s'attend à les trouver sous ce nom dans le répertoire où il est installé.

Le fichier 'SPECEDT.DTA' contient toutes les données

nécessaires à la gestion des boîtes de dialogue (valeurs par défaut, types des contrôles, exclusions, etc.). Enfin le dernier fichier, 'SPECEDT.MNU', sert de support à l'écran d'accueil (nom des commandes et leur classe, identifiant de la boîte de dialogue correspondante). Ces deux fichiers sont écrits en mode binaire et leurs structures de données sont compactées, afin d'assurer la compatibilité avec l'environnement de programmation de Windows.

b. Entrées.

Le programme GEN accepte en entrée deux noms de fichiers: celui contenant la description du langage et celui des ressources. Ces deux noms doivent être corrects au sens MS-DOS du terme, et ne pas dépasser 80 caractères. Si l'un ou les deux noms ne figurent pas dans la ligne de commande faisant appel à GEN, il(s) est (sont) demandé(s) à l'utilisateur. Dans ce cas, le générateur de dialogues propose l'extension '.TPC' par défaut pour le fichier de descriptions et le nom de base du fichier de descriptions avec l'extension '.RC' pour le fichier de ressources. Celui-ci devrait toujours avoir l'extension '.RC', exigée par le compilateur de ressources. Voici un exemple d'appel au programme GEN avec le dialogue qui en résulte. Les mots introduits par l'utilisateur sont imprimés en caractères gras.

Exemple:

```
gen
Fichier de descriptions  .TPC  : file.tpc
Fichier de ressources  FILE.RC  : file.rc
```

Pour accepter le nom proposé par défaut du fichier de ressources, il suffit de taper RETURN en lieu et place d'un autre nom. Dans le cas où l'utilisateur ne donnerait pas le nom du fichier de descriptions, GEN lui redemande le nom du fichier après avoir affiché le message d'erreur:

Le nom du fichier de descriptions est obligatoire

c. Décomposition du programme.

L'architecture du générateur de dialogue a été conçue de manière à refléter la structure des données contenues dans le fichier de descriptions. Celui-ci est censé correspondre à la définition que nous en avons donnée précédemment. Le programme se décompose en plusieurs modules selon une hiérarchie de type "utilise". La plupart de ces modules se rapportent donc à un aspect particulier des données. Nous en exposons ici brièvement l'objet, en invitant le lecteur intéressé par plus de détails à se reporter à l'annexe du présent mémoire. Ainsi, après avoir vérifié que la première ligne du fichier contient le caractère de séparation utilisé avec les valeurs par défaut, le générateur traite ensuite le Command Language Interface (CLI), avec les noms des commandes à afficher dans la fenêtre principale de l'application, pour produire le fichier de menu. Après cela, il lui reste à transformer la suite des commandes avec leurs paramètres en fichiers de ressources et de données.

Dans le traitement proprement dit des commandes, chacune est vue comme une suite, éventuellement vide, de paramètres accom-

pagnés d'informations spécifiques (type prédéfini, valeur par défaut, exclusion, ...). Nous distinguons deux sortes de paramètres: ceux qui seront représentés par une "listbox" et les autres. Cette distinction est rendue nécessaire par le fait qu'une seule "listbox" peut représenter plusieurs paramètres. C'est également la raison pour laquelle il convient de les délimiter dans le fichier de descriptions, comme nous l'avons expliqué.

Les informations relatives aux paramètres sont traitées successivement, dans l'ordre où elles doivent apparaître dans le fichier de descriptions: combinaison de deux paramètres, type prédéfini du contrôle à générer ("listbox", "checkbox" ou bouton radio), caractère obligatoire du paramètre, valeur par défaut et enfin exclusions.

Le paramètre lui-même est transformé en un ou plusieurs contrôles, dont le nombre et le type sont déterminés par l'analyse d'une ou plusieurs lignes du fichier de descriptions, en respectant le schéma du type visuel approprié.

Tous les contrôles générés doivent transiter par un fichier temporaire, car ce n'est qu'à la fin de la commande que le programme peut connaître la hauteur de la boîte de dialogue qu'il vient de créer. Cette valeur, rappelons-le, doit figurer sur la première ligne de définition d'une boîte de dialogue. Il dispose alors de toutes les informations nécessaires et peut recopier le contenu du fichier temporaire à sa place, entre les lignes rela-

tives à la boîte de dialogue en tant que telle (titre, identifiant, style, etc) et celles qui décrivent les trois boutons poussoirs ('Ok', 'Cancel' et 'Help'), placés tout en bas de la boîte.

Afin de faciliter notre tâche, nous avons conçu quelques fonctions "utilitaires". Il s'agit d'une fonction de lecture dans un fichier texte, permettant une tentative de récupération en cas d'erreur, d'une fonction chargée de composer les différents types de contrôles (fenêtre d'édition, texte, "checkbox", bouton radio et "listbox") et enfin de deux fonctions de décryptage des informations relatives respectivement aux valeurs par défaut et aux exclusions.

d. Erreurs.

Au cours de l'exécution du générateur de dialogues, certaines conditions d'erreur sont susceptibles de se présenter. Elles sont de deux types. Ce sont d'abord des erreurs de manipulation d'un fichier, qui, bien que peu probables, peuvent survenir à n'importe quel moment. Elles provoquent toujours l'arrêt prématuré du programme. Une deuxième catégorie d'erreurs concerne le fichier de description du langage. Exception faite de la présence d'une commande ne figurant pas dans le Command Language Interface, elles sont toutes fatales. Dans tous les cas, l'utilisateur est averti de l'incident par un message que nous avons voulu le plus explicite possible.

Ainsi les messages d'erreur relatifs aux fichiers indiquent tous de quel genre d'erreur il s'agit et mentionnent le fichier défaillant. De même, ceux qui sont provoqués par une interprétation erronnée du fichier de description du langage affichent toujours la ligne du fichier incriminée et le type d'information que le générateur de dialogues s'attendait à y trouver.

## C. L'éditeur de commandes.

### a. Ecran d'accueil.

L'écran d'accueil de l'éditeur de commandes SPECEDT présente initialement une barre de titre, une barre de menu et une fenêtre vide.

La barre de titre sert à indiquer le nom du programme et son mode de fonctionnement actuel. Lors du lancement d'une session de travail, c'est le mode interactif qui est sélectionné par défaut. La barre de titre apparaît alors comme dans la Figure 6.

Lorsque le mode batch est choisi, cette information y est aussi répercutée, par l'affichage du nom du fichier de commandes utilisé. La barre de titre sera par exemple celle de la Figure 7.

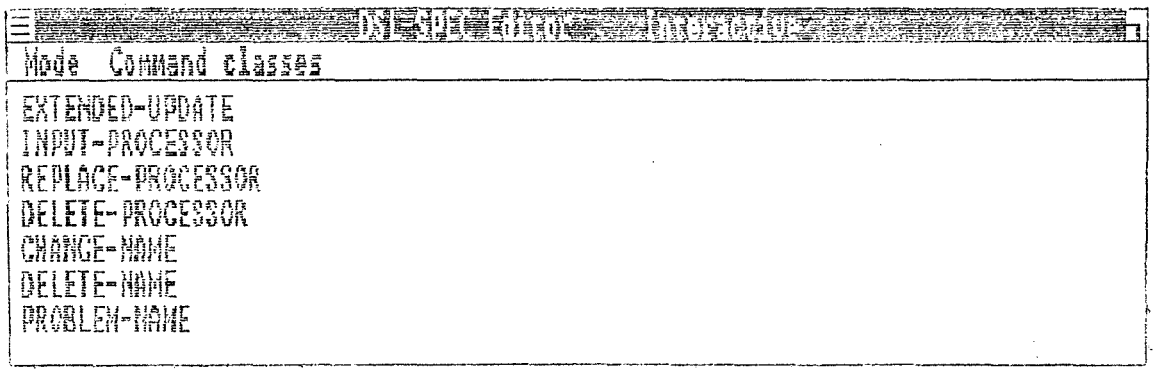



Figure 6.

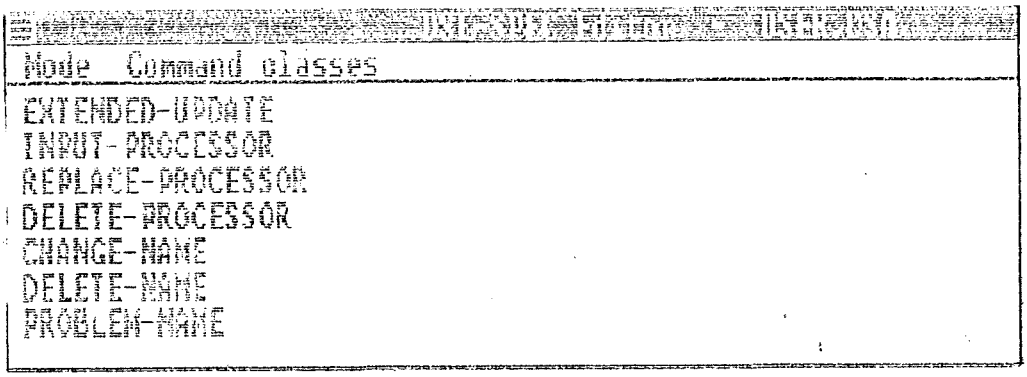



Figure 7.

La barre de menu quant à elle, permet de choisir le mode d'utilisation de SPECEDT, et de sélectionner la liste des commandes à afficher dans la fenêtre.

Les deux modes d'utilisation possibles sont le mode interactif, dans lequel les commandes sont envoyées sur le site central par l'intermédiaire d'un émulateur de terminal, et le mode batch, où les commandes sont écrites dans un fichier. Le mode de fonctionnement actuellement sélectionné est signalé de deux manières à l'utilisateur: d'abord dans la barre de titre, et puis par le fait qu'il ne peut plus être sélectionné dans la barre de menu, où il apparaît en grisé.

Lorsque l'utilisateur souhaite créer un fichier de commandes, il doit opter pour le mode batch. Une boîte de dialogue apparaît alors, qui lui demande de choisir un fichier, éventuellement dans un répertoire ou sur un lecteur différent. Cette boîte de dialogue comprend une fenêtre d'édition, permettant d'afficher un nom par défaut, d'introduire directement un nom de fichier et

de visualiser celui qui est actuellement sélectionné, une "list-box" avec tous les noms de fichiers existants ayant la même extension que le nom par défaut ainsi que les sous-répertoires et lecteurs accessibles, une fenêtre de texte montrant le nom du répertoire actuel, et enfin deux boutons poussoirs Open et Cancel servant respectivement à accepter le fichier sélectionné et à annuler le choix de l'option batch (voir Figure 8). L'utilisateur peut alors introduire le nom de son fichier dans la fenêtre d'édition ou le sélectionner dans la "listbox". Pour changer de lecteur ou de répertoire, il lui suffit de double-cliquer sur le nom correspondant dans la "listbox". Lorsque son choix est fait, il n'a plus qu'à cliquer sur le bouton Open. Si l'utilisateur a choisi un fichier inexistant, l'éditeur de commandes lui demande, par une boîte de message, s'il veut créer un nouveau fichier ou non (voir Figure 9). S'il répond YES, un nouveau fichier est créé dont le nom apparaîtra désormais dans la barre de titre de l'application. Sinon, il se retrouvera avec la boîte de dialogue afin de choisir un autre nom.

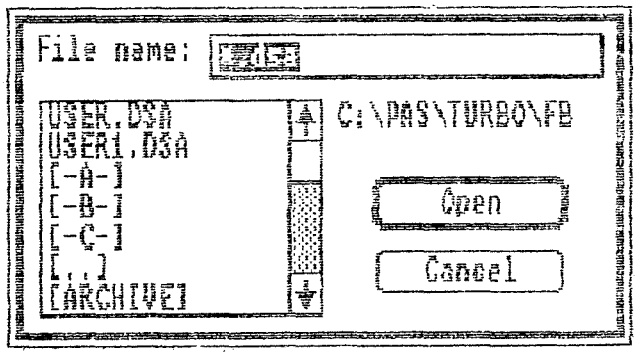



Figure 8.

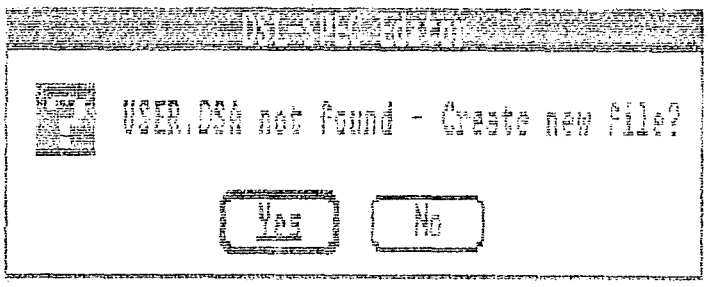



Figure 9.

Le mode batch commence obligatoirement par la commande SET, qui est invoquée automatiquement.

Pour le quitter et revenir en mode interactif, l'utilisateur dispose de plusieurs possibilités. Tout d'abord, au lieu de choisir un nom à son fichier, il peut cliquer sur le bouton poussoir Cancel. Ensuite, à un moment quelconque de sa session de travail, il peut sélectionner et envoyer la commande STOP. Celle-ci, après avoir été copiée dans le fichier de commandes, provoque le retour au mode interactif. Parallèlement, le choix du mode interactif dans la barre de menu de la fenêtre principale a pour conséquence l'envoi de la commande STOP dans le fichier batch.

L'affichage de la liste des commandes disponibles se fait au moyen du menu déroulant 'Command classes'. Celui-ci indique toutes les classes de commandes disponibles, ainsi que l'option 'All' qui permet de les sélectionner toutes pour l'affichage. L'utilisateur peut donc, selon son choix, faire apparaître à l'écran l'ensemble des commandes ou seulement celles appartenant à une classe déterminée. Les commandes apparaîtront alors dans la

zone cliente de l'éditeur de commandes et l'option choisie sera mise en évidence par un "check mark" devant son libellé.

Si les commandes ne peuvent toutes tenir sur la surface de la fenêtre, une barre de défilement horizontale apparaît. Celle-ci permet de faire défiler la liste des commandes horizontalement, vers la gauche ou vers la droite, colonne par colonne ou page par page, ou bien encore de visualiser la liste à partir d'un endroit particulier (voir Figure 4, page 51).

Lorsque l'utilisateur clique sur le nom d'une commande, celle-ci est sélectionnée et elle apparaît en "vidéo inverse" (noir sur fond blanc).

Tout comme dans la version précédente de l'interface, il y a deux moyens d'éditer une commande: soit en faisant appel à la boîte de dialogue correspondante, dans laquelle apparaissent tous les paramètres et leur valeur par défaut, soit en introduisant directement la commande dans une fenêtre d'édition. L'accès à la boîte de dialogue complète se fait en double-cliquant sur le bouton gauche de la souris, tandis que la boîte de dialogue générique destinée à recevoir directement la commande est invoquée par un double-clic sur le bouton droit de la souris. Cette manière de procéder permet à l'utilisateur ayant acquis une certaine connaissance du langage de réduire fortement le temps d'édition de la commande.

b. Boîtes de dialogue.

Chaque boîte de dialogue servant à l'édition des commandes comporte, comme nous l'avons décrit lors de la conception de l'interface, une barre de titre avec le nom de la commande, une barre de menu, un ensemble de contrôles représentant les différents paramètres et trois boutons poussoirs 'Ok', 'Cancel' et 'Help'.

La barre de menu permet d'enregistrer les valeurs par défaut de l'utilisateur dans un fichier, d'initialiser un seul ou tous les contrôles de la boîte de dialogue avec les valeurs ainsi mémorisées ou avec celles du système, et enfin de visualiser la commande en cours d'édition, telle qu'elle sera envoyée dans le fichier batch ou à l'émulateur de terminal.

En sélectionnant cette dernière option, l'utilisateur fait apparaître une nouvelle fenêtre sur la boîte de dialogue, en surimpression des trois boutons poussoirs (voir Figure 5, p. 58). Dans cette fenêtre est affichée la commande avec tous les paramètres dont la valeur actuelle est différente de la valeur par défaut du système. Lorsqu'elles existent, les formes abrégées sont toujours utilisées, tant pour le nom de la commande que pour ceux des paramètres ou de leurs valeurs, puisque l'un des objectifs de cette fenêtre est de familiariser l'utilisateur novice avec le langage de commande et ses abréviations.

La commande éditée contient une ou plusieurs lignes, selon le format accepté sur le site central: chaque ligne, à l'exception de la dernière, se termine par un blanc suivi du signe "-" et aucune ne contient plus de 80 caractères; les paramètres y sont séparés par un blanc et aucun blanc ne peut figurer dans un paramètre.

Le menu déroulant 'Get default' permet à l'utilisateur de réinitialiser soit un seul contrôle de la boîte de dialogue, le dernier auquel il a accédé, soit tous, et ce soit avec les valeurs par défaut reconnues par le système, soit avec ses propres valeurs qu'il doit avoir mémorisées au préalable avec la commande de la barre de menu 'Set default'. Par défaut, ce sont les options 'System' et 'All' qui sont utilisées; elles sont d'ailleurs précédées d'un "check mark" lorsque la boîte de dialogue apparaît à l'écran. Toute option de ce menu sélectionnée par l'utilisateur est marquée de la même manière, et le signe devant l'option complémentaire est effacé.

Lorsque la boîte de dialogue apparaît, tous les contrôles se voient assigner leur valeur par défaut, s'il y en a une, c'est-à-dire que les "checkboxes" et boutons radio seront éventuellement positionnés, que les chaînes de caractères figurant dans les "listboxes" y seront copiées et la valeur par défaut sélectionnée, et, dans le cas des fenêtres d'édition, elle y sera affichée. L'utilisateur pourra alors, selon les besoins de sa tâche ou au

gré de sa fantaisie, modifier la valeur de ces contrôles, parmi lesquels il peut se déplacer soit avec la souris, soit avec le clavier en frappant la touche TAB pour avancer et SHIFT-TAB pour remonter la liste. Comme il a été prévu de le faire, toute action dans une fenêtre d'édition s'accompagne du (dé)positionnement du bouton radio ou de la "checkbox" qui lui est associé, s'il y en a un. Dès que l'utilisateur quitte une fenêtre d'édition, son contenu est immédiatement vérifié s'il y a lieu de le faire. SPECEDT le prévient alors d'erreurs éventuelles par un message approprié, dans une boîte de message ayant pour titre 'Syntax error' (voir Figure 10). Les différents messages affichés lors de la vérification syntaxique des fenêtres d'édition sont les suivants:

Integer value too small for parameter nom

Integer value too large for parameter nom

Not an integer value for parameter nom

Invalid first character 'car' in DSL name for parameter nom

Invalid character 'car' in DSL name for parameter nom

DSL name too long in parameter nom

Too many names in list parameter nom

(où nom est le nom du paramétre et car celui du caractère dont la valeur a provoqué le message d'erreur).

L'utilisateur peut quitter la boîte de dialogue en cliquant sur l'un des boutons Ok et Cancel. Dans le premier cas, la syntaxe de la commande est verifiée et SPECEDT affiche le cas

échéant un message d'erreur dans la boîte de message que nous venons de décrire. Les deux messages qui ne peuvent apparaître qu'à ce moment sont les suivants:

No value for mandatory parameter nom

No value for checked parameter nom

Lorsque toutes les valeurs introduites pour les paramètres sont acceptables, et que tous les paramètres positionnés ou obligatoires ont une valeur, la boîte de dialogue disparaît de l'écran et la liste des commandes est de nouveau affichée. La commande venant d'être éditée est alors envoyée dans le fichier batch ou à l'émulateur de terminal.

Pour l'instant, la fonction d'aide, accessible grâce au bouton poussoir 'Help', n'est pas implémentée. Toute action sur ce bouton reste donc sans effet.

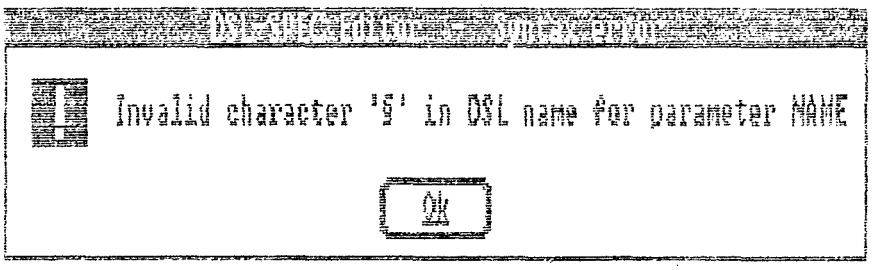



Figure 10.

c. Décomposition du programme.

L'éditeur de commande DSL-SPEC a, comme les autres programmes tournant sous l'environnement Windows, une architecture un peu particulière. En effet, hormis la fonction principale et les fonctions d'initialisation du programme, toutes les autres sont appelées en réponse à certains messages du système. Nous n'en reprendrons ici que les principes essentiels. Le lecteur avide de plus de détails les trouvera en annexe.

Ces messages s'adressent, en général, à une fenêtre particulière, et lui font part, entre autres, des actions de l'utilisateur, comme par exemple la frappe d'une touche du clavier ou un clic de la souris à tel ou tel endroit. La fonction de gestion de la fenêtre décide alors si le message doit être traité et, le cas échéant, déclenche l'action appropriée.

Les fonctions implémentées ne sont donc activées qu'à l'occasion de messages déterminés dans certaines fenêtres ou boîtes de dialogue: c'est le cas, par exemple, de la fonction chargée d'envoyer la commande à l'émulateur du terminal ou de l'écrire dans le fichier de commandes lorsque l'utilisateur clique sur le bouton poussoir 'Ok' ou frappe la touche 'RETURN'.

Il faut remarquer que les messages peuvent se produire en grande quantité et à des fréquences assez élevées, en fonction notamment de l'activité de l'utilisateur. De plus, le très haut niveau d'interactivité offert par l'environnement permet de passer sans transition d'une application à l'autre.

Toutes ces caractéristiques impliquent un fort degré de modularisation des programmes et ont des conséquences, par exemple, sur la gestion des fichiers: ici, il ne peut être question de laisser un fichier ouvert quand le programme rend la main au système entre deux messages. C'est grâce à cela que, pour prendre un exemple qui nous concerne, l'utilisateur aura la possibilité de consulter (ou modifier) le fichier batch, constamment mis à jour, chaque fois qu'il y aura envoyé une commande.

De même, les fonctions activées en réponse aux actions de l'utilisateur doivent, idéalement, avoir un temps d'exécution fort court, de manière à lui apporter un "feedback" le plus rapidement possible.

Voici, brièvement exposés, les principaux ingrédients qui entrent dans la composition d'un "bon" programme pour l'environnement Windows.

# CONCLUSION

Arrivé au terme de ce travail, et bien que les propositions visant à rendre plus conviviale l'interface de l'éditeur de commandes DSL-SPEC dans l'environnement Windows n'aient pu toutes être réalisées, nous pensons qu'il est temps de jeter un coup d'oeil en arrière et d'évaluer le chemin parcouru.

Nous allons donc reprendre les principales critiques que nous avions formulées dans la première partie de ce mémoire, pour voir si nous y avons répondu et de quelle manière les solutions que nous y avons apportées contribuent à l'amélioration de la qualité de cette interface.

Un premier point concernait la sous-utilisation de l'espace disponible. Il ne s'agit pas à proprement parler d'un défaut de l'interface, car un écran peu chargé est toujours préférable à un autre où les informations apparaissent comme un vaste agglomérat indéchiffrable de symboles. Le problème résidait plutôt dans le fait qu'il faille sans cesse faire défiler cet écran. Nous pensons que la présentation des différentes commandes du langage sous forme de formulaires dans des boîtes de dialogue, telle que nous l'avons implémentée, constitue une solution fort intéressante, d'autant plus qu'elle va de pair avec l'utilisation de différentes facilités, comme par exemple l'utilisation équivalente du clavier et de la souris, définies dans l'environnement de programmation dans le but, justement, d'améliorer la qualité des interfaces.

Il convient cependant de ne pas trop pavoiser, car le risque est grand, si le langage était étendu et de nouveaux paramètres ajoutés à certaines commandes, d'aboutir assez vite à des écrans à ce point surchargés que leur lisibilité en serait dangereusement compromise. Si cette éventualité se présentait, il faudrait alors modifier la solution proposée, par exemple en utilisant deux (niveaux de) boîtes de dialogue pour une même commande.

En supprimant les boîtes de dialogue intermédiaires dans la définition des paramètres, nous avons supprimé une indirection qui n'était sans doute pas vraiment souhaitable, et par là le temps nécessaire à l'édition d'une commande s'en trouve réduit, ce qui est fort probablement une bonne chose. Mais cette amèlioration n'est pas gratuite: le prix à payer est la perte d'une possibilité de fournir à l'utilisateur des informations relatives tant au type qu'au format de la valeur qu'il va introduire, et qui étaient incluses dans ces boîtes de dialogue intermédiaires. Cette perte d'information n'est que partiellement compensée par la variété des types visuels que nous avons définis, avec, entre autres, l'utilisation de fenêtres d'édition dont la taille varie selon le type d'information à y introduire.

C'est pourquoi une fonction d'aide indiquant, au minimum, le type et le format des valeurs à attribuer aux paramètres nous avait paru si importante.

Quant à la liste des commandes du langage qui est affichée dans la fenêtre principale de l'éditeur, son défilement est rendu beaucoup moins fréquent non seulement par le fait de la présenter en plusieurs colonnes, mais aussi grâce à la répartition des commandes en plusieurs classes, formant ainsi autant de sous-menus. Cette subdivision de la liste des commandes a été perçue par toutes les personnes que nous avons interrogées à ce sujet comme une amélioration indispensable.

Dans le même ordre d'idées, le fait de pouvoir faire défiler cette liste de n'importe quelle manière, tant avec le clavier qu'à l'aide de la souris, nous a paru combler une lacune très gênante dans la première version de l'éditeur de commandes.

Un dernier point de l'interface que nous tenions à améliorer est la récupération des erreurs, ce qui, à notre avis, a été réalisé d'une manière tout à fait acceptable, même si la qualité des messages d'erreur pourrait encore être perfectionnée.

Enfin, l'addition de quelques fonctionnalités à l'éditeur de commandes, par exemple la fenêtre de visualisation ou le fait de pouvoir accéder à tout moment aux valeurs par défaut que l'utilisateur s'est lui-même définies, est sans conteste un pas en avant vers la définition d'une interface conviviale et aussi facile et agréable à utiliser pour un expert du langage de commande que pour quelqu'un qui y fait ses premiers pas.

Nous pensons ainsi avoir réussi à montrer, par un exemple concret, que l'utilisation judicieuse des facilités offertes dans l'environnement de programmation de Windows, basée sur quelque décisions de conception relativement simples mais efficaces, permettait d'accroître de façon appréciable le caractère ergonomique et convivial d'une classe particulière de programmes.

REFERENCES.

Ben Shneiderman: "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison-Wesley, 1987.

David Durant, Geta Carlson, Paul Yao: "Programmer's Guide to Windows", SYBEX, 1987.

Manuel de référence DSL-STATION, METSI, janvier 1988.

Manuel de référence DSL-SPEC, Rel. 2.0R0, mars 1986.

Guide d'utilisation de DSL-SPEC dans un environnement VAX/VMS, version IDA2R0, novembre 1986.

Microsoft C Compiler: User's Guide, Language Reference, Run-Time Library Reference, CodeView Window-Oriented Degugger, Microsoft Corporation, 1984-1986.

Microsoft Windows: Programming's Utility Guide, Programming Guide, Programmer's Reference, Application Style Guide, Microsoft Corporation, 1984-1986.

# UN GENERATEUR DE DIALOGUES SOUS MS-WINDOWS. ANNEXES.

François Bouhon
3° Licence Informatique

# UN GENERATEUR DE DIALOGUES SOUS MS-WINDOWS.

# (ANNEXES)

François Bouhon
3° Licence Informatique

Année Académique 1987-1988

PLAN.

## I. ARCHITECTURE DU PROGRAMME GEN.

Le générateur de dialogues a été conçu de manière à refléter les données contenues dans le fichier de descriptions, ce qui en facilite grandement la modularisation sur base d'une hiérarchie de type "utilise". Cette décomposition se justifie de la façon suivante. Chacun des modules a trait à un aspect particulier du problème et peut à ce titre être considéré comme un sous-système utile du programme: ses composantes s'exécutent de manière autonome par rapport au reste du programme, et elles présentent effectivement des fonctionnalités relatives à un sous-problème.

Dans le but de clarifier le comportement du programme, nous exposons également la découpe des modules en fonctions. Celles-ci, tout comme les modules, s'articulent autour d'une hiérarchie de type "utilise".

Afin de faciliter notre tâche, nous avons conçu quelques fonctions "utilitaires". Il s'agit d'une fonction de lecture dans un fichier texte, 'lire', permettant une tentative de récupération en cas d'erreur, d'une fonction, 'createcontrol', chargée de composer les différents types de contrôles (fenêtre d'édition, texte, checkbox, bouton radio et listbox) et enfin de deux fonctions, 'trtdef' et 'trtexcl', traitant respectivement les valeurs par défaut, et les exclusions. Bien que ces fonctions soient totalement indépendantes les unes des autres et n'aient pas de point commun entre elles, nous nous permettons de les présenter comme

faisant partie d'un même "module", UTILITAIRES, ceci afin de simplifier les schémas des pages suivantes. Dans ceux-ci, le nom des fonctions apparaît en minuscules et celui des modules en majuscules. Les fonctions des différents modules accessibles de l'extérieur sont soulignées.

## 1. Architecture logique.

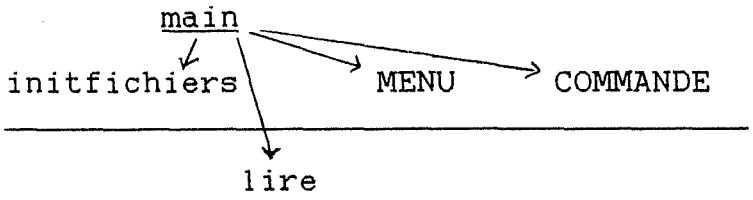

UTILITAIRES

Le module MAIN est le coordinateur du programme. C'est lui qui doit détecter les différentes parties du fichier de descriptions et utiliser le module approprié pour le traitement de chacune d'elles: soit le module MENU pour produire le fichier de menu à partir de la description du Command Language Interface, soit le module COMMANDE pour chaque description de commande qui sera transformée en ressources et données appropriées.

Ce dernier module utilise les deux modules LIST et PARAM pour la définition respectivement des contrôles "listbox" et des autres.

Finalement, chacun d'eux utilise une ou plusieurs des fonctions regroupées dans le module UTILITAIRES.

## 2. Module MAIN.



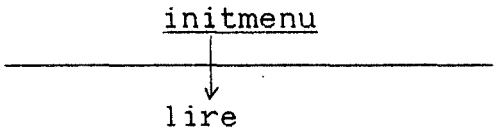

Tout d'abord, la fonction principale 'main' utilise obligatoirement la fonction 'initfichiers' pour traiter les arguments éventuels passés au programme dans la ligne de commande et pour l'ouverture des différents fichiers. Ces arguments sont les noms de deux fichiers: celui contenant la description syntaxique du langage, et celui qui sera produit par le programme de génération avec les définitions des boîtes de dialogue de l'interface. Si l'un au moins de ces deux noms est absent de la ligne de commande, il est demandé à l'utilisateur. Le dialogue qui s'ensuit avec l'utilisateur en vue de l'acquisition par le programme du ou des noms de fichier qui n'ont pas été donnés en argument fait partie intégrante du traitement desdits arguments.

Vient ensuite la génération proprement dite de l'interface pour l'éditeur de commandes SPECEDT. Après avoir vérifié la présence du caractère de séparation, le programme passe la main au module MENU pour extraire les données du Command Language Interface (CLI) et produire le fichier de menu. Après cela, il lui reste à transformer, par appels répétés au module COMMANDE, la suite des commandes avec leurs paramètres en boîtes de dialogue et

leurs contrôles décrits à la fois dans le fichier de ressources et celui de données.

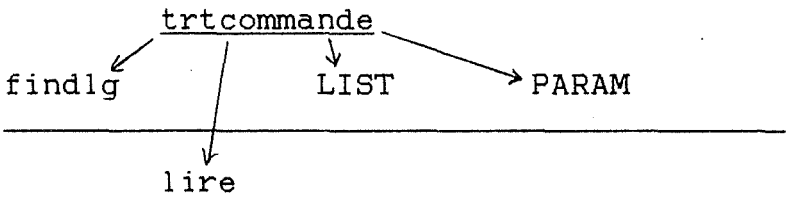
La fonction utilitaire 'lire' est employée chaque fois qu'il s'agit de lire une ligne d'un fichier texte, comme d'ailleurs dans tout le reste du programme.

## 3. Module MENU.




Le module MENU correspond à la fonction 'initmenu', dont le but est le garnissage du fichier de menu, ouvert et initialement vide, à partir de la "pseudo-commande" CLI du fichier de description. Le générateur suppose que la dernière ligne lue dans le fichier de description contient le mot-clé 'CLI', et que la liste des commandes à lire se termine par la carte 999. La fonction 'initmenu' utilise la fonction 'lire' pour la lecture ligne par ligne du fichier de description.

## 4. Module COMMANDE.



C'est le module COMMANDE qui a la charge de produire la définition de la boîte de dialogue et la structure de données correspondant à la commande en cours de traitement.

Celle-ci est vue comme une suite de paramètres accompagnés d'informations spécifiques (type prédéfini, valeur par défaut, exclusion, ...). Nous distinguons deux sortes de paramètres: ceux qui seront représentés par une "listbox" et les autres. Cette distinction est rendue nécessaire par le fait que les "listboxes" peuvent représenter plusieurs paramètres. C'est égamement la raison pour laquelle il convient de les délimiter dans le fichier de descriptions.

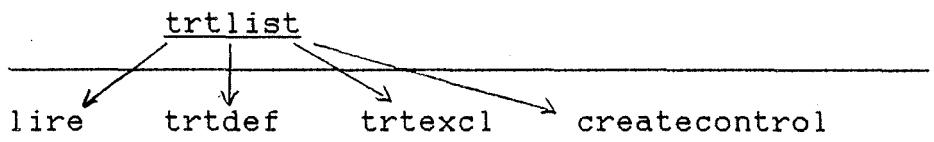
Les informations relatives aux paramètres sont traitées successivement, dans l'ordre où elles doivent apparaître dans le fichier de descriptions: combinaison de deux paramètres, type prédéfini du contrôle à générer ("listbox", "checkbox" ou bouton radio), caractère obligatoire du paramètre, valeur par défaut et enfin exclusions.

Le paramètre lui-même est transformé en un ou plusieurs contrôles, dont le nombre et le type sont déterminés d'après l'analyse d'une ou plusieurs lignes du fichier de descriptions et selon le schéma des types visuels que nous avons définis. Cette transformation se fera soit dans le module LIST, soit dans le module PARAM, selon qu'il s'agit de produire une "listbox" ou non.

Tous les contrôles générés doivent transiter par un fichier temporaire, car ce n'est qu'à la fin de la commande que le programme peut connaître la hauteur de la boîte de dialogue qu'il vient de créer. Il dispose alors de toutes les informations nécessaires et peut recopier le contenu du fichier temporaire à sa place, entre les lignes relatives à la boîte de dialogue en tant que telle (titre, identifiant, style, etc) et celles qui décrivent les trois boutons (Ok, Cancel et Help), placés tout en bas de la boîte. C'est la fonction 'findlg' qui est chargée de copier ces informations dans le fichier de ressources.

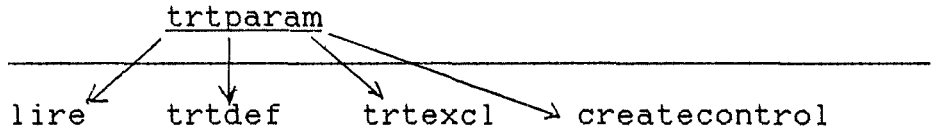
Tout comme dans la fonction principale et le module MENU, il est fait appel à la fonction 'lire' chaque fois qu'une lecture d'un fichier texte s'impose.

## 5. Module LIST.




C'est la fonction 'trtlist' qui produit les contrôles "listbox" et les données associées. Elle fait appel aux quatres fonctions utilitaires pour traiter des aspects particuliers du problème.

## 6. Module PARAM.




C'est la fonction 'trtparam' qui produit tous les autres types de contrôles et les données associées. Elle fait appel aux quatres fonctions utilitaires pour traiter des aspects particuliers du problème. Les contrôles sont produits suivant le schéma des types visuels que nous avons définis.

## 7. Fonctions utilitaires.

La fonction 'lire', ainsi que nous l'avons déjà exposé, est utilisée pour lire une ligne dans un fichier texte ouvert, dont le nom lui est communiqué.

La fonction 'trtdef' sert à mémoriser dans la structure de données la valeur par défaut figurant dans la dernière ligne lue du fichier de description.

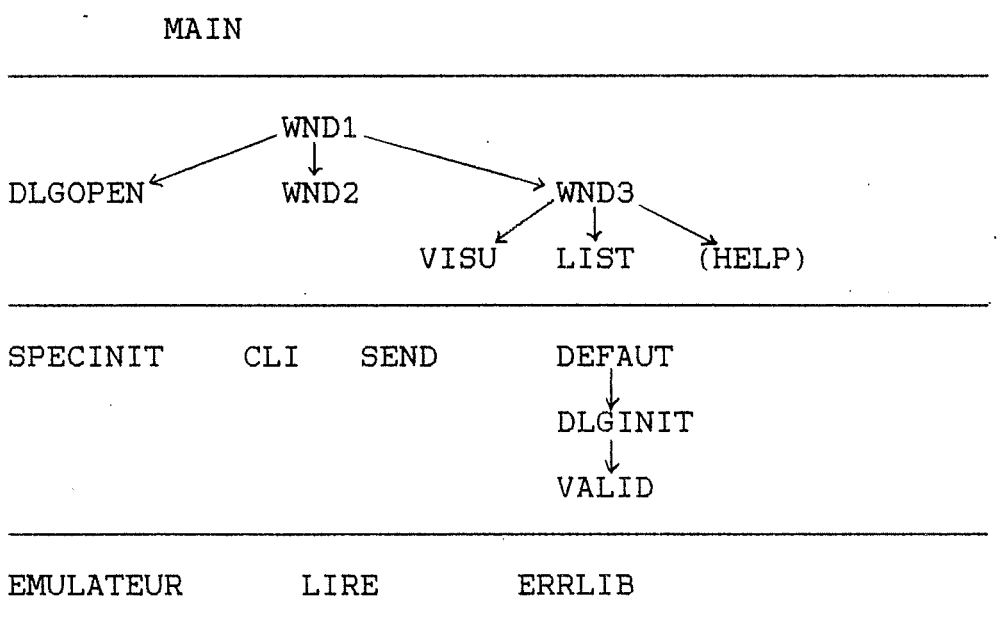
La fonction 'trtexcl' trouve son origine dans le fait qu'il faut transformer les identifiants des paramètres, tels qu'ils sont dans le fichier de description, en identifiants des contrôles correspondants.

Quant à la fonction 'createcontrol', elle est invoquée chaque fois qu'il faut produire la définition d'un contrôle de type donné. Elle garnit également les champs d'identifiants de la structure de données.

## II. ARCHITECTURE DU PROGRAMME SPECEDT.

L'objet de cette annexe est de détailler quelque peu, par vues progressives, l'architecture de l'éditeur de commandes SPECEDT. Certains modules ont été récupérés de la version précédente de l'interface: ce sont les modules servant à communiquer avec l'émulateur de terminal (EMULATEUR), à produire des boîtes de message (ERRLIB) et à acquérir un nom de fichier par l'intermédiaire d'une boîte de dialogue (DLGOPEN). Nous utilisons les mêmes conventions de notation que pour l'architecture du programme GEN (voir Annexe I).

### 1. Architecture logique.



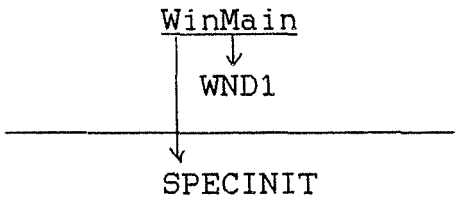

L'architecture logique de l'éditeur de commandes se divise en quatre couches de modules: le coordinateur du programme (module

MAIN), les modules de gestion des différentes fenêtres et boîtes de dialogue, ceux qui traitent les différentes données de l'application, et enfin quelques modules utilitaires.

Il est évident que la hiérarchie sous-jacente à cette répartition en quatre couches est de type "utilise". En effet, la gestion de chaque type de fenêtre apparaît comme un sous-système utile du programme. De même, les différent modules de traitement des données répondent à autant de sous-problèmes distincts de la gestion des fenêtres et de l'application en général. Finalement, les modules utilitaires sont utilisés pour répondre à certains besoins précis et bien délimités de l'application, parfois dans des contextes fort différents (comme par exemple le module ERRLIB).

A l'intérieur de chaque couche de l'architecture logique, il peut également exister une hiérarchie de type "utilise" entre certains modules. Dans le cas de la gestion des fenêtres, elle s'identifie à la hiérarchie "parentale" des fenêtres. Les relations entre modules utilisateurs et utilisés apparaîtront sans doute de manière plus claire dans la suite, lorsque nous détaillerons un peu plus chaque cas particulier. Nous ne parlerons pas ici des trois modules déjà définis dans la précédente interface et que nous avons réutilisé (DLGOPEN, EMULATEUR et ERRLIB).

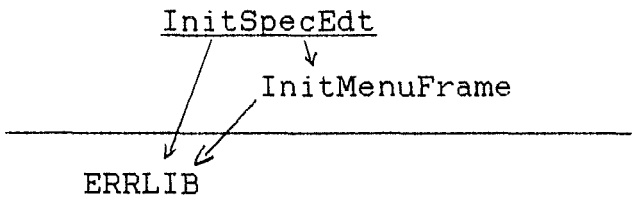
## 2. Module MAIN.




La fonction principale, 'WinMain', est le centre même du programme, du moins dans sa partie visible. Il faut en effet remarquer que, si tous les messages envoyés par et aux différentes fenêtres passent par cette fonction, le système cache une portion considérable de leur prise en charge et leur expédition vers la fonction de traitement appropriée.

'WinMain' fait d'abord appel au module SPECINIT pour effectuer les initialisations nécessaires au lancement de l'application, avant d'entrer dans la boucle de traitement des messages traditionnelle sous Windows. Ces messages sont passés aux fonctions de gestion des fenêtres, en particulier au module WND1 pour tout ce qui concerne l'écran d'accueil.

## 3. Module SPECINIT.

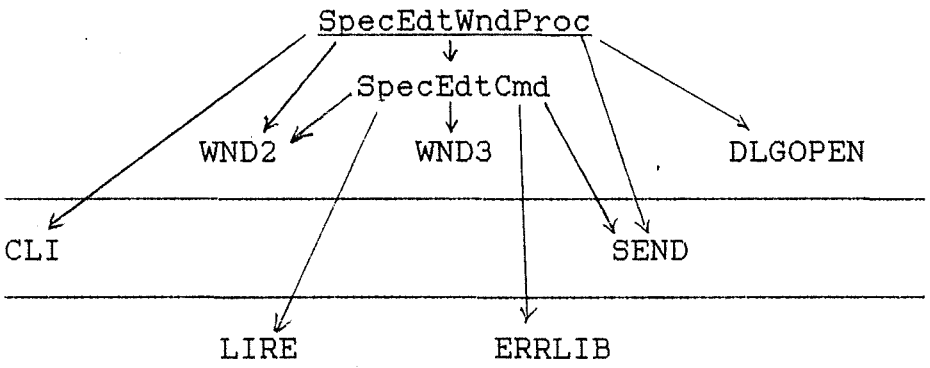



Ce module regroupe les deux fonctions utilisées chaque fois que l'éditeur de commandes est lancé.

La fonction 'InitSpecEdt' s'occupe de toutes les initialisations communes à la plupart des applications sous Windows (enregistrement de la classe de fenêtres, création de la fenêtre principale, ...), ainsi que de l'initialisation des variables servant à l'affichage de la liste des commandes. Elle utilise la fonction 'InitMenuFrame' pour garnir la structure de menu à partir du fichier de menu.

Le module ERRLIB est invoqué pour les messages d'erreur.

## 4. Module WND1.




Le module WND1 contient les deux fonctions principales de gestion de l'écran d'accueil. Celles-ci font appel à d'autres modules pour en traiter des aspects particuliers.

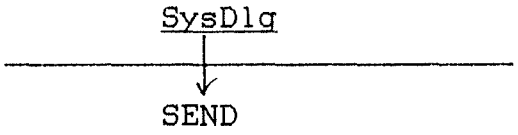
La fonction 'SpecEdtWndProc' utilise le module CLI pour tout ce qui concerne l'affichage de la liste des commandes, et le module DLGOPEN pour l'acquisition d'un nom de fichier batch.

La fonction 'SpecEdtCmd' est invoquée lorsque l'utilisateur manifeste son désir d'éditer une commande. Elle utilise le module LIRE pour accéder aux données du fichier de données, le

module WND3 pour la gestion de la boîte de dialogue correspondant à la commande à éditer et le module ERRLIB en cas d'erreur.

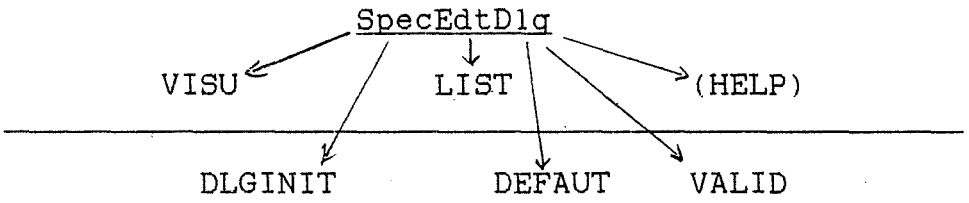
Les deux fonctions utilisent le module SEND pour envoyer une commande, et le module WND2 pour gérer la boîte de dialogue permettant l'introduction directe d'une commande.

## 5. Module WND2.



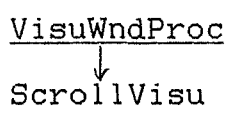

Ce module comprend la fonction 'SysDlg', chargée de la gestion de la boîte de dialogue servant à l'introduction directe d'une commande ou pour la commande SYSTEM. Il utilise le module SEND pour envoyer la commande.

## 6. Module WND3.



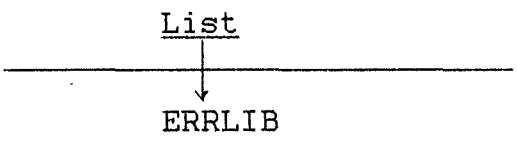

La fonction 'SpecEdtDlg' gère les boîtes de dialogue servant à l'édition des commandes. Pour ce faire, elle utilise les modules DLGINIT, DEFAUT, VISU, VALID et LIST. Le module HELP, correspondant à la fonction d'aide, devra venir s'y insérer.

## 7. Module VISU.



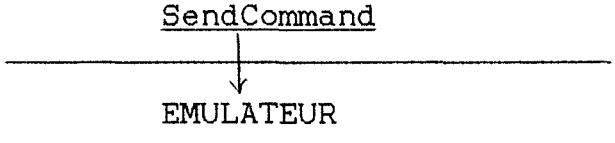

Ce module contient les fonctions 'VisuWndProc', qui a la charge de gérer la fenêtre de visualisation, et 'ScrollVisu', invoquée en cas de demande de défilement de cette fenêtre.

## 8. Module LIST.




C'est le module chargé de gérer les boîtes de dialogue de second niveau, qui présentent les listes de valeurs associées aux paramètres RELATIONS et STATEMENT. En cas d'erreur, il utilise le module ERRLIB pour l'affichage d'un message.

## 9. Module SEND.



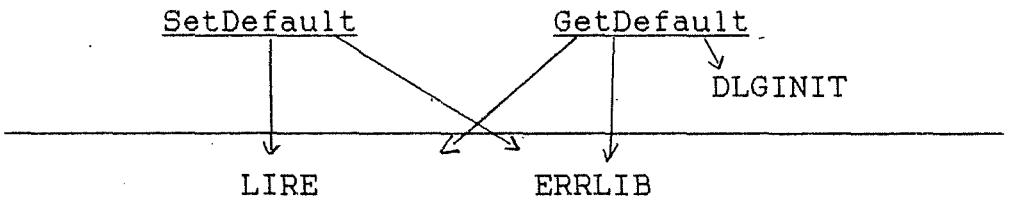

Il contient la fonction 'SendCommand', utilisée pour envoyer une commande éditée dans le fichier batch ou à l'émulateur de terminal. Dans ce dernier cas, le module EMULATEUR est utilisé.

## 10. Module CLI.

ScrollCLI
↓
AfficherCLI
↓
ecrire

Ce module, qui regroupe les fonctions utilisées pour l'affichage de la liste des commandes, a deux points d'entrée possibles: soit la fonction 'ScrollCLI', soit directement la fonction 'AfficherCLI', utilisées respectivement pour faire défiler la fenêtre principale et pour l'affichage de la liste des commandes. La fonction utilitaire 'ecrire' est employée par 'AfficherCLI' pour afficher une chaîne de caractères à l'écran.

## 11. Module DEFAUT.




Les deux fonctions 'SetDefault' et 'GetDefault' sont indépendantes l'une de l'autre. Elles sont néanmoins regroupées au sein du même module puisqu'elles traitent du même aspect de l'interface: les valeurs par défaut. Toutes deux utilisent les modules LIRE, pour l'acquisition des données, et ERRLIB, en cas d'erreur. De plus, 'GetDefault' fait appel au module DLGINIT pour réinitialiser les contrôles de la boîte de dialogue.

12. Module DLGINIT.

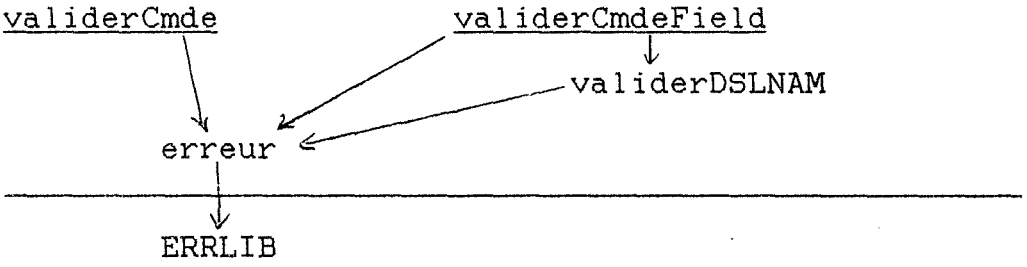
dlginitfield ComposerCmde modifierdlg

↓

VALID

Ce module contient toutes les fonctions propres aux contrôles des boîtes de dialogue et qui utilisent la structure de données du fichier SPECEDT.DTA: initialisation, modification et composition de la commande. Ces fonctions sont indépendantes l'une de l'autre. En cas de modification au sein d'une fenêtre d'édition, il est fait appel au module de validation VALID.

13. Module VALID.




C'est dans ce module que sont rassemblées les différentes fonctions destinées à vérifier la correction syntaxique de la commande en cours d'édition.

La fonction 'validerCmde' est invoquée pour s'assurer que les paramètres obligatoires et ceux qui ont été positionnés ("checkboxes" ou boutons radio) ont effectivement une valeur.

La fonction 'validerCmdeField' vérifie la validité syntaxique des valeurs introduites dans les fenêtres d'édition. Elle

utilise la fonction 'validerDSLNAM' dans le cas d'un nom ou d'une liste de noms d'objets DSL.

Chacune de ces fonctions, lorsqu'une erreur est détectée, fait appel à la fonction 'erreur' pour préparer le message d'erreur approprié qui sera communiqué à l'utilisateur par l'intermédiaire du module ERRLIB.

## 14. Module LIRE.

Il est composé de la fonction 'LireDonnees', qui recherche dans un fichier une structure de données correspondant à l'identifiant indiqué, que ce soit celle des défauts de l'utilisateur ou celle contenant toutes les informations nécessaires aux boîtes de dialogue, et la mémorise à l'endroit indiqué. Si la structure n'est pas trouvée ou si une erreur de manipulation du fichier se produit, une variable globale est positionnée, qui indiquera un code d'erreur.